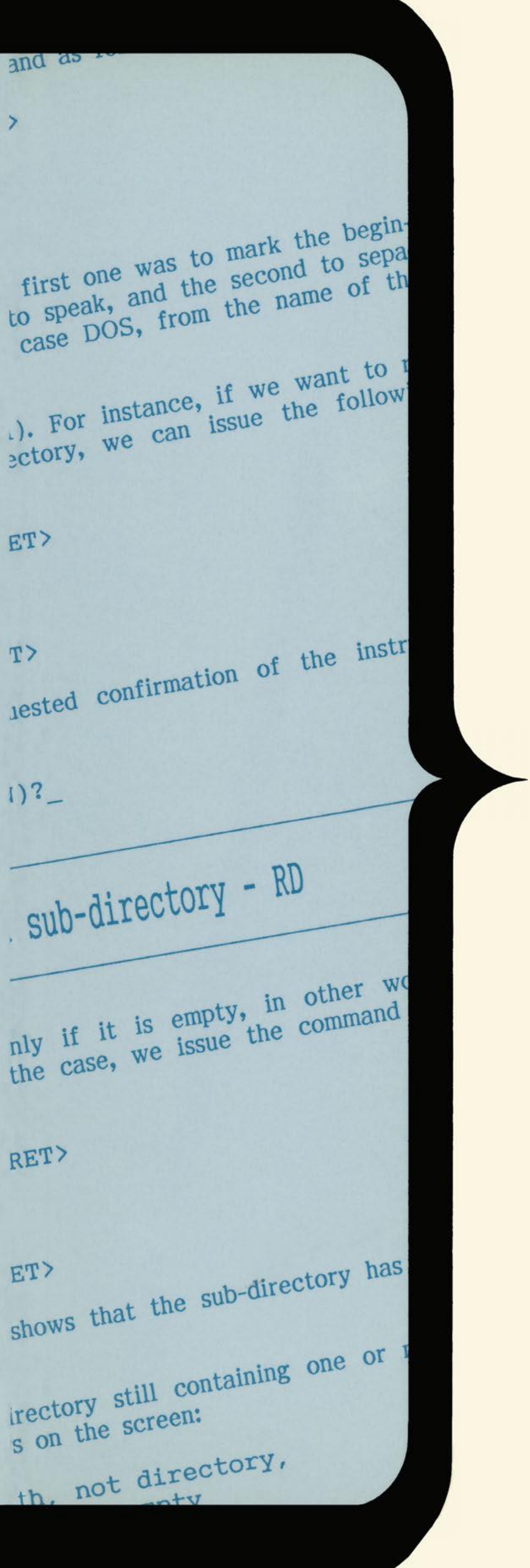

MS-DOS and PC-DOS
A Practical Guide

Pim Oets

Second Edition

Macmillan Computer Science Series
Consulting Editor
Professor F. H. Sumner, University of Manchester

S. T. Allworth and R. N. Zobel, *Introduction to Real-time Software Design, second edition*
Ian O. Angell and Gareth Griffith, *High-resolution Computer Graphics Using FORTRAN 77*
Ian O. Angell and Gareth Griffith, *High-resolution Computer Graphics Using Pascal*
M. Azmoodeh, *Abstract Data Types and Algorithms*
C. Bamford and P. Curran, *Data Structures, Files and Databases*
Philip Barker, *Author Languages for CAL*
A. N. Barrett and A. L. Mackay, *Spatial Structure and the Microcomputer*
R. E. Berry, B. A. E. Meekings and M. D. Soren, *A Book on C, second edition*
G. M. Birtwistle, *Discrete Event Modelling on Simula*
B. G. Blundell, C. N. Daskalakis, N. A. E. Heyes and T. P. Hopkins, *An Introductory Guide to Silvar Lisco and HILO Simulators*
B. G. Blundell and C. N. Daskalakis, *Using and Administering an Apollo Network*
T. B. Boffey, *Graph Theory in Operations Research*
Richard Bornat, *Understanding and Writing Compilers*
Linda E. M. Brackenbury, *Design of VLSI Systems–A Practical Introduction*
J. K. Buckle, *Software Configuration Management*
W. D. Burnham and A. R. Hall, *Prolog Programming and Applications*
J. C. Cluley, *Interfacing to Microprocessors*
J. C. Cluley, *Introduction to Low Level Programming for Microprocessors*
Robert Cole, *Computer Communications, second edition*
Derek Coleman, *A Structured Programming Approach to Data*
Andrew J. T. Colin, *Fundamentals of Computer Science*
Andrew J. T. Colin, *Programming and Problem-solving in Algol 68*
S. M. Deen, *Fundamentals of Data Base Systems*
S. M. Deen, *Principles and Practice of Database Systems*
Tim Denvir, *Introduction to Discrete Mathematics for Software Engineering*
P. M. Dew and K. R. James, *Introduction to Numerical Computation in Pascal*
M. R. M. Dunsmuir and G. J. Davies, *Programming the UNIX System*
D. England et al., *A Sun User's Guide*
K. C. E. Gee, *Introduction to Local Area Computer Networks*
J. B. Gosling, *Design of Arithmetic Units for Digital Computers*
M. G. Hartley, M. Healey and P. G. Depledge, *Mini and Microcomputer Systems*
Roger Hutty, *Z80 Assembly Language Programming for Students*
Roland N. Ibbett, *The Architecture of High Performance Computers*
Patrick Jaulent, *The 68000–Hardware and Software*
P. Jaulent, L. Baticle and P. Pillot, *68020-30 Microprocessors and their Coprocessors*
J. M. King and J. P. Pardoe, *Program Design Using JSP–A Practical Introduction*
E. V. Krishnamurthy, *Introductory Theory of Computer Science*
V. P. Lane, *Security of Computer Based Information Systems*
Graham Lee, *From Hardware to Software–an introduction to computers*

continued overleaf

A. M. Lister and R. D. Eager, *Fundamentals of Operating Systems, fourth edition*
Tom Manns and Michael Coleman, *Software Quality Assurance*
G. P. McKeown and V. J. Rayward-Smith, *Mathematics for Computing*
Brian Meek, *Fortran, PL/1 and the Algols*
A. Mével and T. Guéguen, *Smalltalk-80*
Barry Morrell and Peter Whittle, *CP/M 80 Programmer's Guide*
Derrick Morris, *System Programming Based on the PDP11*
Y. Nishinuma and R. Espesser, *UNIX–First contact*
Pim Oets, *MS-DOS and PC-DOS–A Practical Guide, second edition*
Christian Queinnec, *LISP*
E. J. Redfern, *Introduction to Pascal for Computational Mathematics*
Gordon Reece, *Microcomputer Modelling by Finite Differences*
W. P. Salman, O. Tisserand and B. Toulout, *FORTH*
L. E. Scales, *Introduction to Non-Linear Optimization*
Peter S. Sell, *Expert Systems–A Practical Introduction*
A. G. Sutcliffe, *Human-Computer Interface Design*
Colin J. Theaker and Graham R. Brookes, *A Practical Course on Operating Systems*
M. R. Tolhurst et al., *Open Systems Interconnection*
J-M. Trio, *8086-8088 Architecture and Programming*
M. J. Usher, *Information Theory for Information Technologists*
B. S. Walker, *Understanding Microprocessors*
Peter J. L. Wallis, *Portable Programming*
Colin Walls, *Programming Dedicated Microprocessors*
I. R. Wilson and A. M. Addyman, *A Practical Introduction to Pascal–with BS6192, second edition*

Non-series
Roy Anderson, *Management, Information Systems and Computers*
I. O. Angell, *Advanced Graphics with the IBM Personal Computer*
J. E. Bingham and G. W. P. Davies, *A Handbook of Systems Analysis, second edition*
J. E. Bingham and G. W. P. Davies, *Planning for Data Communications*
B. V. Cordingley and D. Chamund, *Advanced BASIC Scientific Subroutines*
N. Frude, *A Guide to SPSS/PC+*
Barry Thomas, *A PostScript Cookbook*

MS-DOS and PC-DOS

A Practical Guide

Pim Oets

Second Edition

M
MACMILLAN

First edition 1985
Second edition 1988

Authorised English Language translation and revision of
Handleiding MS-DOS/PC-DOS, Third Edition, 1987
by Pim Oets
published by Uitgeverij Pim Oets, 1001 ZD Amsterdam,
The Netherlands

Translated by Arlette Eype

Published by
MACMILLAN EDUCATION LTD
Houndmills, Basingstoke, Hampshire RG21 2XS
and London
Companies and representatives throughout the world

British Library Cataloguing in Publication Data

Oets, Pim
MS-DOS and PC-DOS : a practical guide.
2nd ed.—(Macmillan computer science series).
1. IBM PC microcomputer systems. Operating systems : PC-DOS
2. Microcomputer systems. Operating systems : MS-DOS
I. Title
005.4'469

ISBN 978-0-333-45440-4 ISBN 978-1-349-09893-4 (eBook)
DOI 10.1007/978-1-349-09893-4

CONTENTS

INTRODUCTION

This book is designed to meet two different purposes. First, it aims to lend a helping hand to those who have just begun to use a computer under the control of MS-DOS or PC-DOS. Second, it is intended to be a useful source of information and reference for advanced users. As a result, the book can only be a compromise in certain respects.

When the text is intended to help beginners find their way through the jungle of terms and commands, step-by-step explanations are given, beginning with the simplest operations. In addition, the importance of not over-estimating the background knowledge of the beginner has been continually kept in mind. Clear examples, straightforward definitions of all terms and names, and where necessary copious cross-references are given throughout the text.

On the other hand, when the text is intended to provide reference material for the more advanced user, the various commands and procedures are discussed in much greater detail, including lesser-known facilities and applications. In addition, a clear table of contents and detailed indexes are provided.

In order to meet these aims, it was decided to divide the book into chapters dealing with a specific subject or topic. For instance, chapter 3 comprises explanations and commands that mainly refer to creating and manipulating files; this chapter is therefore called Files.

By reading each chapter from beginning to end, the beginner can easily become familiar with the most important terms and procedures with regard to the particular topic. The advanced user may in turn consult the table of contents or the index to find the chapter or section in which a certain term or command is discussed. To meet this goal, and also for some other practical reasons, it was necessary to ignore some procedures and commands that are mainly relevant to programmers.

The various types, makes and configurations of the computer systems under MS-DOS and PC-DOS provide another problem. For instance, finding appropriate examples or terms for explanatory remarks becomes complicated because of the differences between hard disks and floppy disks.

To solve this problem, we have chosen the following method of working. In the first four chapters, the examples are based on the presumed presence of two floppy disk drives, and from the fifth chapter, they are based on a floppy disk and a hard disk. This is not relevant to beginners who have a computer with two floppy disk drives. They may skip chapter 5 until they are ready to deal with chapter 6, when they will presumably be familiar enough with the subject matter to overcome the difference in drives.

If the beginner owns a computer with a hard disk, some problems may be met when going through the first four chapters. By reading chapter 5 first, these problems can easily be resolved.

Another problem is the plethora of different versions of MS-DOS and PC-DOS that have been put on the market over the past years. This is a result of both the technical as well as the commercial history of the operating system. In short, it boils down to the following.

It all began in 1976. The American microprocessor manufacturer Intel Corporation introduced its newest and most advanced product, the 8086 chip. A few years later, the 8086 was joined by a somewhat simpler offspring, the 8088.

The next phase began in the late seventies. Probably stimulated by the success of the Apple II, the giant computer corporation IBM decided to develop its own type of microcomputer. The Intel 8088 was chosen as the heart of the machine. The company approached the new company Microsoft which was mainly concentrating on software for microcomputers and working very successfully on its adjustments to the popular programming language BASIC.

In its turn Microsoft contacted a small company named Seattle Computer Products which had already completed a prototype of an operating system for the 8086 and 8088 chips, naming them 86-DOS and SCP-DOS respectively. Microsoft obtained the rights to the system and hired Tim Paterson, the actual designer of the system, for a period of one year. IBM agreed to this.

When the IBM Personal Computer came on the market, its operating system turned out to bear the name IBM Personal Computer DOS, or DOS for short, the version being 1.00. Although the contents, structure and name of PC-DOS were owned by IBM, Microsoft had reserved the right to supply third parties, in particular the manufacturers of so-called compatibles, with similar versions of the original system of Tim Paterson.

These varieties are collectively known as MS-DOS (for Microsoft DOS). Although some computer manufacturers add their own set of utilities, the compatible PC-DOS and MS-DOS versions in fact function similarly.

Seven new versions have been introduced since the original version of PC-DOS appeared, each of which has been considerably improved and extended with respect to the preceding version. The development and introduction of the various MS-DOS versions coincided with these events.

The first revised PC-DOS version was launched in the summer of 1982 and named 1.10. The main change was that it supported double-sided disk drives. Later that year, the first revised version of MS-DOS, numbered 1.25, was introduced. It was almost identical to the PC-DOS 1.10.

Coinciding with the launching in March 1983 of the IBM PC XT (for Extended), a PC with a built-in hard disk, PC-DOS version 2.00, was introduced. It included a large number of new features. In the autumn of 1983, together with IBM's PC Jr, a further improved version, PC-DOS 2.10, came into circulation, soon to be followed by the almost identical MS-DOS version 2.11.

In the summer of 1984 IBM then introduced a new type of computer, the PC AT (for Advanced Technology), which was based on a new microprocessor, the Intel 80286. As might be expected, a new version of the operating system was simultaneously released named PC-DOS 3.00, and subsequently, in the spring of 1985 a further advanced version, PC-DOS 3.10, which included all the necessary facilities for use in networks.

The sixth version was the PC-DOS version 3.20, shortly afterwards replaced by 3.21. The latter was put on the market in March 1986 on the occasion of the introduction of the PC Convertible. The main novelty of that computer was its small size of disk drive, namely 3.5 inches, with a capacity of 720 Kb. This new version of PC-DOS had especially been designed for this. In the meantime, MS-DOS versions 3.20 and 3.21 were launched as well.

The latest version of PC-DOS is 3.30. It came into existence in April 1987, along with an entirely new series of IBM microcomputers: the PS/2 range. By the end of that same year, Compaq Computer Corporation introduced its 386/20 series, accompanied by version 3.31 of Compaq's brand of MS-DOS. And that is the situation at the time of writing.

Microsoft and IBM have been wise enough to make all the successive versions of the operating system 'upward compatible'. This means that each version contains all the elements of its preceding versions.

In PC-DOS 3.30, for instance, all successively developed facilities of the versions 1.00, 1.10, 2.00, 2.10, 3.00, 3.10 and 3.20/3.21 have been included, improved if necessary, and regularly supplemented with new elements. Likewise, MS-DOS 3.31 includes the facilities collected from all its preceding versions.

In other words, if a certain version of the system is suitable for a certain type of computer, all the preceding versions will be suitable as well. However, this does not apply to the latest versions. Indiscriminately using different versions may even lead to damage to files and should therefore be avoided.

The principle of 'upward compatibility' has made it possible to base this book on some sort of common denominator of all these versions. This results in an emphasis on the MS-DOS versions 2.11 and 3.10, and the PC-DOS versions 3.00 and 3.10. Owners of older versions, for instance PC-DOS 1.10 or MS-DOS 1.25, will certainly find some useful information in this book, in particular in the first four chapters, but the rest of this book is not specifically meant for them.

Chapters 1 to 9 are useful for owners of the 2.00 version or higher. If a certain facility has only been introduced in one of the level-3 versions, this fact is explicitly stated. Chapter 10 deals with the improvements and additions that have only been introduced with versions 3.20, 3.21, 3.30 and 3.31 of MS-DOS and PC-DOS.

Chapter 10, as its title suggests, not only deals with new, but also with variant commands of the system. This implies that there are some minor differences between the various versions. Although it is impractical to list them all, let alone discuss them, there are many general points to make. Where it is of particular interest, new and/or different versions will be discussed in more detail.

The so-called 'error messages' have been assembled in the final chapter of this book, in alphabetical order and with a few comments.

Further to the above, it is appropriate to caution the beginner.

Even under the control of modern operating systems like PC-DOS or MS-DOS, much can go wrong. To begin with, technical failures may occur. Therefore, it is vital that the environment is not subjected to fluctuations, that dust, smoke, magnetic and electrostatic objects are excluded, and that the equipment is not subjected to knocks, shocks or vibrations.

In the second place, we as users may be at fault. Fortunately, the operating system has been subjected to many tests during its development, therefore the chances of mistakes being the system's fault are remote. However, every user has to pass on commands and lines of text repeatedly, and a typing error is easily made: we may omit an essential character or strike the wrong key.

Therefore, it must be borne in mind that we should not skip a single step and that each letter, each number and each punctuation mark, including spaces, have a certain significance to the computer which is not necessarily the same as in common speech. For, as we will see on the first page of the first chapter, a computer is a dumb thing, merely consisting of electronics, screws and air. It does not have a brain that can sense what we may have meant.

If this was not so, there would have been no reason to write this book.

1: OPERATING THE COMPUTER

According to experts, a computer is an electronic instrument capable of executing programs of arithmetical and logical operations without human intervention. However, when we switch on a computer, at first nothing in particular seems to happen. After a few seconds a message may appear on the monitor screen, but tapping keys has no effect at all. Why is this?

To understand the reason, we must realise that a computer is no more than a machine - a mindless device full of screws, wires, switches and air. It is able to perform all kinds of intricate operations, provided that we prescribe step by step what it has to do and that we use the right codes. It requires specialised knowledge as well as an enormous amount of work to feed the necessary instructions into the computer yourself. Therefore these instructions are available in fixed sets, ready for use.

In computer jargon, such a set of instructions is known as an Operating System (OS), or a Disk Operating System (DOS).

Which operating system a manufacturer chooses for his computer depends on various factors, but in the first place on the desired method for storing data. Until the middle of the seventies, data was stored on punch cards or paper slips. This did not change until the American company IBM introduced a method to store data on disks by means of electromagnetism - the present disks or floppy disks.

The result of this new storing method was that the operating systems had to be extended by instructions and procedures for the use of such disks and their accommodation, the so-called disk drives. Consequently, the operating system became a Disk Operating System.

Not only the disk drives, but the other parts and accessories of the computer as well, should be able to be switched on and used. We may think of the keyboard, a mouse, the monitor, a printer or a plotter, a joystick for those who love games, a modem, etc.

In the second place, the choice of the operating system depends on the built-in microprocessor, known as the chip: the miniature electronic circuit, operating as CPU (Central Processing Unit).

In August 1981, when IBM introduced the Personal Computer, abbreviated to PC, the company turned out to have used a chip that was known as Intel 8088. IBM offered no less than three different operating systems for this computer but the cheapest one was finally chosen, probably because the operating systems had to be acquired separately. This became IBM's own Personal Computer DOS, or in short, PC-DOS - version 1.00.

This first IBM PC-model was followed by numerous new versions and types. Moreover, time and again new or improved facilities were developed. These developments resulted in new, improved and extended versions of PC-DOS.

At the same time, attracted by the success of the PC, other manufacturers put similar equipment, known as compatibles, on the market. Together with the developers of PC-DOS, the American software company Microsoft made an arrangement for a similar version of the operating system. This resulted in a counterpart of the PC-DOS which was called MS-DOS, for Microsoft-DOS.

In Chapter 10 (page 169) we will discuss the results of this muddle of versions as well as the similarities and differences we may meet.

1.1: Loading the system (1)

The operating system is usually supplied on a disk which we recognise by a label stating DOS among other things.

In order to operate the computer, some of the data on the system disk has to be put into the computer's memory. In other words, we have to load the system. We start this procedure by placing the system disk in drive A.

The entrance of a disk drive is a horizontal or a vertical slot that we usually find next to or under the monitor. If there is only one slot, it is automatically drive A. If there are two or more, however, drive A is the one on the left or the one on the top. The way we put the disk in the slot is of vital importance. The label has to point in our direction and has to be on the left in a vertical drive and on top in a horizontal drive.

Provided that we hold the disk between the thumb and forefinger of the right hand, with the thumb on the label, we can hardly go wrong.

The next step is to turn on the machine. The switch is usually located on the side or back of the machine.

After switching on - with IBM computers it takes some time - we usually hear a bell or a beep. A number of mysterious lines should then appear on the screen, followed by a text such as:

```
Microsoft MS-DOS version 3.21
Copyright 1981, 1986 Microsoft Corp.
```

If the computer is not equipped with a calendar and clock installation, we then receive the request to enter the date:

```
Current date is Tue 1-01-1980
Enter new date: _
```

How to do this will be described on page 20. We will skip that for now and strike the key marked RETURN or ENTER or showing a broken arrow pointing to the left. We shall refer to this key as the RETURN key. Also see page 16.

Then we are asked to enter the time in the same way:

```
Current time is 0:02:10.39
Enter new time: _
```

How to enter the time will also be described on page 20, but for now we just strike the RETURN key.

If we are working with PC-DOS, the IBM version of the operating system, we see the date and time messages first, and then the message about the identity. For instance:

```
The IBM Personal Computer DOS
Version 3.20 (C)Copyright International Business
             (C)Copyright Microsoft Corp 1981, 1
```

Anyhow, in each case we will see the following symbol:

```
A>_
```

The operating system has now been loaded. The A with a bracket (or, on some models, a colon) is known as the prompt. It is an invitation to act.

The underline (or the small block) is the cursor. It marks the place on the screen where our input will be shown.

It may happen that when we try to load the system, either nothing appears on the screen or we see the following message:

```
Non-System disk or disk error
Replace and strike any key when ready
```

This is a so-called error message, in this case meaning that the computer has not been able to recognise the contents on the disk as the system.

There may be four reasons for the error message:

- maybe we put the disk in the drive incorrectly;
- or maybe we put in the wrong disk. In both cases we should try again.
- maybe the machine is not suitable for this version of the system;
- maybe there is something wrong with the disk itself. In these two cases we should consult the dealer.

What will happen when we turn on the machine when there is no system disk in drive A depends on the brand, type and design.

If we have a computer with a so-called hard disk or fixed disk on which the system has been placed (how to do this will be explained on pages 44 and 81), then we see a prompt with a different letter, usually a C. Thus:

```
C>_
```

This letter is the symbol for the hard disk. For the rest, this prompt functions in the same way as a prompt with an A or any other letter.

If we work with an IBM computer that does not have a hard disk, and we start the computer without a system disk in drive A, we are automatically confronted with the computer language BASIC, which is built into the machine. This is shown by the following message:

```
The IBM Personal Computer BASIC
Version C3.00 Copyright IBM Corp. 1981, 1985
65536 Bytes free
Ok
```

This is a leftover from the old times of the computer, when it was not a matter of course that disk drives were present. A cassette recorder had to be used as a medium of storage. The first models of the IBM PC were therefore equipped with a special means of access for it. Hence the letter C (for Cassette) that follows the word Version in the message.

If we do not want to work with Cassette BASIC, we have to put a system disk in drive A after all and start a RESET procedure (see page 18), or we have to turn the machine off and turn it on again.

1.2: The keyboard (1)

As soon as the operating system has been loaded, we can use the keyboard as if we were sitting in front of an electronic typewriter. The position of the keys is practically the same. It is referred to as QWERTY, after the first six letter keys in the second row.

Let's type something. For instance:

```
A>qwerty <RET>
```

In this book, underlining (as above) will be used to indicate what must be typed on the keyboard to have it appear on the screen.

A term between angle brackets < > indicates further action to be taken. So, <RET> means that we should press the RETURN key. This is the key marked with the term RETURN or ENTER, or with a broken arrow pointing to the left.

The result of pressing this key is that whatever has been typed is passed on to the processor. If it has any meaning, it will be processed. In this example this is obviously not the case, for the system replies with the following error message:

```
Bad command or file name
```

Let's have a closer look at the keyboard. The keys indicated by SHIFT, or showing a hollow arrow pointing up, are the keys to type capital letters. They usually can be found at the left- and right-hand side of the row of letters right above the space bar.

Most computers working under MS-DOS or PC-DOS have a block of keys with the figures 0 to 9, plus a number of arithmetic symbols, just like a calculator, at the right-hand side of the keyboard. This keypad is intended to simplify typing figures. To do so, however, we should first press the NUM LOCK key.

If we do not press the NUM LOCK key, the keys of the block of figures have a different function. With certain programs they can be used to steer the cursor. We can identify them by the terms HOME, END, PG UP (for Page Up), PG DN (for Page Down) and four keys showing arrows pointing left, right, up and down.

In addition to the above-mentioned keys we find various other special keys; for instance, the key marked SCROLL LOCK and/or BREAK, ESC (for Escape), CTRL (for Control), ALT (for Alternate), TAB (for Tabulate), INS (for Insert) and DEL (for Delete). We will come across a number of these special keys further on in this book.

On the far left or along the top of the keyboard, there are a number of keys marked F1, F2 etc. In a certain operating system or other program these keys may be given a certain prescribed function - hence the indication 'function keys'. See also page 22.

Finally, IBM computers and most compatibles are equipped with a special key marked PRT SC (which means Print Screen). If we strike this key together with one of the SHIFT keys, the printer will produce a copy of the characters that are on the screen at that moment, assuming that the printer is connected to the computer and turned on.

1.3: Commands

Loading the operating system means that it is being stored in the computer's memory. Consequently, any valid command issued by us will be processed by the system.

A command is a code that is transmitted to the central processing unit of the computer by means of the system. We may compare this unit to a switching panel with the keys on the keyboard acting as the controls. It receives the characters we type, checks the order and then converts them into instructions.

The operating system is also known as an interface. This term may be best described as an electronic gateway. The prompt offers us access to this gateway. It indicates the line on the screen where we are to type our command. This is the command line. The system cannot process more than 128 characters on the command line, i.e one line and a half. So the maximum length of a command is 128 characters, including spaces.

When we have finished typing a command, we pass it on by striking the RETURN key. The ingenious thing about the system is that the commands are combinations of key strokes which form English verbs or concepts, such as DATE, FORMAT or DISKCOPY. This helps us to memorise the commands.

For the computer, however, these key strokes only mean an instruction to throw a few switches.

For some commands, the system disk is no longer needed in drive A after having loaded the system, as the following experiment will prove. We take the disk out of drive A and we type:

```
A>b: <RET>
```

As we see, the result is that the prompt is no longer an A, but a B and a bracket pointing to the right. Thus:

```
B>_
```

Now B has become the active drive instead of A. In other words: the drive to which our commands will apply from now on.

We reverse this situation as follows:

```
B>a: <RET>

A>_
```

Note that a colon directly follows the letter of the drive to be activated.

If the computer is equipped with a hard disk, we usually have to state this by means of the letter C. Thus:

```
A>c: <RET>

C>_
```

1.4: Correcting a mistake

How do we remove meaningless characters from the command line, or in general, typing errors?

The first way, as we saw on page 16, is to pass them on to the computer by tapping the RETURN key and then wait and see what happens. Thus:

```
A>qwerty <RET>
Bad command or file name

A>_
```

Although the system usually replies with the above error message and a new prompt, the characters may have a meaning for the computer and will start a certain procedure or mess up a program in operation. So this method is rather risky.

The second method is to adopt a RESET procedure. It consists of striking the CTRL, ALT and DEL keys at the same time. The result is that the memory is emptied and the operating system reloaded.

Some computers have a special RESET button. When we use this button, we get the same result as striking the three keys mentioned above.

However, we should be very cautious with RESET procedures. In this example we are only dealing with a few characters on the screen, but we could - as a result of emptying the memory - lose hours of intensive work.

The third method consists of removing the characters from the screen. To do so we have to use a key on the upper right of the keyboard, showing an arrow pointing to the left. This key can usually be found on the right of the top row of the keyboard. We shall refer to this key as 'backspace key'.

In the absence of this key we can also use another key to be found on the left of the keyboard which is marked CTRL (for Control). By holding it down and striking an ordinary letter key, we give the system a so-called control instruction.

The meaning of the instruction depends on the letter chosen, as we will see on page 70.

An example is the control instruction ^H, or the combination of the CTRL key and the H key. As a result of this instruction the character to the left of the cursor will disappear, in this case the y. Thus:

```
A>qwert_
```

The fourth method is the most practical one: striking the ESC key. A backslash will then be displayed after the characters that have been typed to indicate that the system will ignore the characters in question. Thus:

```
A>qwerty\
_
```

The cursor is now at the beginning of the next line and we may try again.

1.5: Asking for the system version - VER

Now it is time to have a real instruction carried out by the operating system. We can, for instance, let it check the version of the system we are dealing with. Instead of starting a RESET procedure in order to reload the operating system (see page 18), we may type the following, simple command:

```
A>ver <RET>
```

As a result we see something like:

```
MS-DOS Version 3.21

A>_
```

1.6: Asking for the label - VOL

Another simple instruction we can give to the system is to trace down the label of a disk. Such a label can be used to find the contents of the disk in question quickly. To do so we use the term VOL, for Volume. An example:

```
A>vol <RET>

Volume in drive A is SYS(3_21)
```

If the disk in question has no label, we receive the following message:

```
Volume in drive A has no label

A>_
```

1.7: Clearing the screen - CLS

If we want to clear the screen after we have finished our work or inspection, we can do so by using the command CLS, for Clear Screen. Thus:

```
A>cls <RET>
```

The result is that all the characters disappear from the screen as if by magic, while the cursor takes up position in the top left-hand corner.

1.8: Entering and changing date and time (1)

It is practical, but not necessary, to enter the date when loading the system.

```
Current date is Tue 1-01-1980
Enter new date: _
```

Suppose that it is 7 December 1988. This is entered in figures in the following manner: first the month (12), then the day (07), and finally the year (1988), separated by hyphens (-). Thus:

```
Current date is Tue 1-01-1980
Enter new date: 12-07-1988
```

The computer itself will work out that 7 December is a Wednesday. We do not have to add that information. Furthermore, we may leave out the 0 in 07 and the 19 in 1988:

```
Enter new date: 12-7-88
```

Apart from a hyphen, a slash or full stop may be used. If we use the full stop, we type:

```
Enter new date: 12.7.88 <RET>
```

When we have made a typing error and we notice it before we have hit the RETURN key, we can correct this by tapping the backspace key or using the instruction ^H (see page 18).

In case we did not correct the mistake, the system will respond with the following error message:

```
Invalid date
Enter new date: _
```

We then have to type the date again, or we can forget about it and tap the RETURN key.

The time comes next:

```
Current time is 0:02:10.39
Enter new time: _
```

Entering the time is not necessary either, but usually it is practical.

Let's assume that it is 3 minutes past eight in the evening. We enter this in figures. First the hour (20), then the number of minutes (03), and finally the number of seconds, separated by a colon or full stop. We may leave out the hundredths of a second. Thus:

```
Enter new time: 20.03 <RET>
```

The system's visiting card we saw on page 14 will now appear, followed by the prompt.

If we have not entered a date when we loaded the system, or if we want to use a different date for whatever reason at a later stage, then we can change or enter it at any time. In that case we type the DATE command in its bare form. Thus:

```
A>date <RET>
```

We may also use capitals, but typing lower case is of course easier.
In our case the system's answer is:

```
Current date is Wed 12-7-1988
Enter new date: _
```

Note that the day of the week (Wed, for Wednesday) has automatically been added to the date.

We now have the opportunity to enter a new date just as we did on the previous page. If we decide not to and just strike the RETURN key, the date remains unchanged.

To save time, we can combine the command directly with the desired new date, for instance 16 December 1988:

```
A>date 12.16.88 <RET>

A>_
```

The prompt indicates that there have not been any problems and that we may give another command.

If we have not entered the time when we loaded the system, or if we want to enter a different time for whatever reason at a later stage, then we can change or enter it at any time. In that case we start the proceedings as follows:

```
A>time <RET>
Current time is 20:47:56.03
Enter new time: _
```

Note that the hundredths of a second are automatically added to the time after the full stop.

We can now enter a new time, just as we did on the previous page. If we decide not to and just strike the RETURN key, the time remains unchanged.

In order to make things easier, we may combine this command, too, with the desired new time, for instance 5 past 11 in the morning:

```
A>time 11.05 <RET>

A>_
```

1.9: Function keys

Most computers using the MS-DOS or PC-DOS system have a keyboard with a fairly large number of special keys, as we saw on page 16.

Some of these keys can be recognised by a term such as INS or DEL or by arrows pointing left, right, up or down. Others only have a number, such as F1, F2 etc.

All these special keys have a certain, prescribed function but in order to avoid the term 'function keys' causing confusion, we shall call the first type editing keys. See page 104.

The second type, the numbered keys, will be considered the actual function keys. The function of the keys marked F1 to F4 is to reproduce the last command line issued, so that we simply can repeat the command or make corrections in it.

We must imagine that a line of characters, which we have passed on to the computer by striking the RETURN key, is not only processed, in other words will result in a reaction from the computer, but is also temporarily stored in a part of the computer's memory specially reserved for this. This part of the memory is known as a buffer.

Storage in the buffer is ended as soon as we pass on a new command line or after we turn off the computer or when we start a RESET procedure (see page 18).

As long as the command line is stored in the buffer, we can reproduce it on the screen entirely or partially by means of the first four function keys.

An example: assume we have loaded the system, that we have omitted to enter the date and that we have done so at a later stage by using a DATE command (see page 20). We have not typed any commands or taken any other action since then.

When we strike the F3 key, the last command we gave to the system will immediately appear on the screen.

For instance:

```
A>date 12-16-88_
```

If we now tap the RETURN key, the command will be executed as if we have just typed it ourselves, as an experiment will prove.

When we strike the F1 key instead of the F3 key, only the first letter of the command line will appear on the screen:

```
A>d_
```

Hitting the key a second time will provide us with the second letter and so on, until we have enough or until the whole line is on the screen:

```
A>date 12-16-88_
```

The same effect can be achieved by striking the key on the numerical pad marked with an arrow pointing to the right.

The F2 key can be used to reproduce a part of the command line up to a specific place.

Suppose that we appear to have made a mistake; that the date is not 16 but 17 December. We can correct this error in several ways.

The first method is to strike the F3 key and then to tap the backspace key or to type an ^H instruction (see page 18) in order to erase the last four characters from the screen. We then type the correct numeral and finally we tap the F3 key again. The result:

```
A>date 12-17-88_
```

The second method is to tap the F1 key nine times, then to type a 7 and use the F3 key again. Once again we see:

```
A>date 12-17-88_
```

The third method, however, is the most efficient one. If we tap the F2 key immediately followed by a 6, we see, as soon as we release the key, the following on the screen:

```
A>date 12-1_
```

If we now type a 7 and then tap the F3 key, we have only had to tap four keys to correct the error:

```
A>date 12-17-88_
```

Using the F4 key will result in the opposite: the command line is reproduced from a certain point.

To demonstrate, we tap the F4 key immediately followed by a 1. At first nothing appears on the screen, but as soon as we tap the F3 key, we see:

```
A>17-88_
```

This does not seem to be of much use. But suppose we have made a typing error, for instance:

```
A>date 13-17-88 <RET>
```

The system's answer is:

```
Invalid date
Enter new date: _
```

If we now tap the F4 key followed by a 7, and finally use the F3 key, this will all result in the following:

```
Invalid date
Enter new date: 12-17-88_
```

Now we only have to strike the RETURN key.

The longer we operate the system, the more we will need methods to save key strokes. Then we will realise how useful these function keys can be.

2: STORAGE MEDIA

It is characteristic for a computer that it not only can execute a large number of prescribed instructions very rapidly, but also that it can store, temporarily or permanently, such instructions as well as their results and other data. For this purpose, the computer is equipped with various facilities. We will discuss four of these briefly.

The first type of storage forms a permanent part of the machine. It is known as RAM, for Random Access Memory, i.e. the computer's working memory, and consists of a collection of so-called memory chips. Storing a great number of instructions into RAM is part of loading the system (see page 14).

Although RAM is an inextricable part of the computer, its contents are anything but permanent. Everything that has been stored in the working memory will vanish into thin air as soon as we start a RESET procedure (see page 18), or turn off the computer. That is why the operating system has to be reloaded again. Therefore, we should consider RAM as a casual facility.

The second type is also a permanent part of the computer. It is the so-called ROM, for Read-Only Memory. Just like RAM, ROM consists of a collection of chips. Technically speaking, ROM also belongs to the working memory of the computer as the name indicates, but there are vital differences with RAM. Firstly, it is not a volatile facility. It is not a temporary but a permanent storage of instructions which means that it is not affected by a RESET procedure or by turning off the machine. Secondly, we cannot change its contents.

The ROM chips are placed into the computer by the manufacturer. They contain a number of basic instructions and information we always need, such as the form of the letters and figures that are shown on the screen. With the PC's and PS/2's of IBM, the heart of the computer language BASIC is built-in in the memory in a similar way, that is, also by means of the ROM chip (see pages 15 and 155).

All the computer codes and instructions stored in ROM are usually indicated by the term BIOS or ROM BIOS. BIOS is an abbreviation of Basic Input/Output System.

The third type of storage is the disk. A disk is neither a permanent part of the computer, nor a volatile medium. Switching on or off the computer does not influence its contents.

If we want to operate, alter or erase the contents of a disk, we have to take a number of accurately prescribed steps in the right order, as we will see further on in this chapter and in chapter 3.

Finally we distinguish the hard disk, also known as fixed disk, as a storage medium. In fact, this is a combination of the previous facilities.

For a start, the fixed disk, as the name indicates, is a permanent part of the computer just like RAM. In addition, a fixed disk adds extra ROM to the computer's memory containing certain operating instructions. Moreover, the contents of a fixed disk are, just like the floppy disk, not affected by a RESET procedure or switching off the computer.

2.1: Types and sizes

The computer's disk drive in which we insert the disk somewhat reminds us of a record player, and the disk of the record. An ingenious device searches the contents of the disk, just like a record, and transfers the data it finds to another place where it will be interpreted and displayed. A record player usually needs an amplifier and a loudspeaker to reproduce the data. A computer, on the other hand, needs a central processing unit (CPU) and a monitor or a printer.

Another feature of both records and disks is that there is quite a number of different types and sizes. Just as it is possible that a record does not function properly if we mix up the playing speeds, we have to be aware of the similarities and differences of the disks.

For a start, we distinguish three different sizes. In the first place, the classical size, with a diameter of 8 inches (20 cm) which was introduced by IBM in the seventies as a revolutionary novelty. However, we will hardly ever come across this type in personal computers, and therefore it may be left out of consideration.

In the second place, we are dealing with a size of disk that was used over the past years in the major part of PC brands and types, and is consequently considered the standard disk: the flexible disk, better known as the floppy disk, with a diameter of 5.25 inches (about 14 cm). This type is again divided into different types of density. See next page.

In the last place, there is a disk that is not flexible but packed in a small hard plastic box with the practical size of 3.5 inch, that is, nearly 9 cm. This has been developed by the Japanese electronics firm Sony and used for several PC's on a small scale over the past years. However, now that IBM has adopted this size for its entire PS/2 line, it might become the new standard type.

In this chapter, we regard a disk as a floppy disk with a diameter of 5.25 inches. The commands and parameters relating to the 3.5-inch disk in particular will be discussed on page 181.

Apart from similarities between records and disks, there are essential differences. In the first place, the data stored on a record is irrevocably laid down in the material. The only alterations we can insert are scratches and other damage. The contents of a disk, on the other hand, can be erased or replaced by something else, if desired. In this respect, a floppy disk or hard disk can be compared with a sound tape or cassette.

Another difference, equally important though less obvious, is the way in which the data is stored on the surface. The contents of the data on a record are stored in two grooves: on the A side and on the B side. These grooves run in the form of a spiral from the edge to the middle of the record. The contents on a disk are divided into a certain number of circular fields: the so-called tracks. These tracks usually (though not always) take up two sides of the disk: side 0 and side 1.

A disk normally has 40 or 80 tracks per side, numbered 0 to 39, or 0 to 79. These tracks are in turn divided into sectors, sometimes eight, but usually nine or fifteen, numbered from 1 to 9 or 1 to 15.

Each sector has a restricted space for a specific amount of information: usually 512 characters, legible for the computer, or rather 512 bytes (see page 32). A calculation shows that the storage capacity of a normal, double-sided disk containing nine sectors per track amounts to two times forty times nine times 512 bytes, that is, over 360 Kb.

However, dividing the sizes is not the end of it. We are also dealing with the type of drive and with the question of maximum storage capacity of each type of drive.

In the first types of disk drives, only one side of the disk could be filled since it was merely equipped with a single read and write head. In addition, the capacity was rather restricted. This type of disk drive will not be discussed in this book for at the time IBM had put its first PC on the market, the evolution had developed to twice the density. Disks with the standard size (5.25 inches) were suited for this purpose. They are referred to as SSDD disks (for single-sided, double-density).

In the course of 1982, the PC's of IBM and its competitors were equipped with a new type of disk drive which had a double read-and-write head. This new type required standard disks that could contain data on both sides. These disks were called DSDD (for double-sided, double-density).

But it is even more complicated. Dependent on the version of the operating system, a varying number of sectors per track can be added to the disk. Version 1 of the system did not have a choice; the number was eight. The versions 2 and 3 of PC-DOS and MS-DOS were increased to nine, but it remained possible to divide the disk into eight sectors per track.

In 1984 two new types of disk drives with an increased storage capacity were introduced. In the first place, the DSQD-type (for double-sided, quad-density), which could contain four times the capacity to a maximum of 720 Kb, and in the second place, the HD type (for High-Density) or HC (for High Capacity), with a storage capacity of no less than 1.2 Mb.

Disks with 80 instead of 40 tracks could be used in these two types of disk drives. Moreover, in the HC-type the number of sectors per track was increased from 9 to 15. In these two types of disk drives, the normal disks can be read, but it is not reliable to fill them. In order to use the maximum storage capacity, special (and more expensive) HC-disks are needed.

This last type of disk drive was chosen by IBM to become a new type of PC, the so-called PC AT, which was put on the market in the autumn of 1984. As a result of the success of this type, as well as of the compatible computers based on this type, it has in fact become a new standard.

Looking at the types of 5.25-inch disks, the different types of drives, and the maximum storage capacities, we can compile the following survey:

	SS drive		DS drive		QD drive	HC drive
	8 sect.	9 sect.	8 sect.	9 sect.		
SSDD disk	160 Kb	180 Kb	160 Kb	180 Kb	160 Kb	180 Kb
DSDD disk	160 Kb	180 Kb	320 Kb	360 Kb	320 Kb	360 Kb
DSQD disk	160 Kb	180 Kb	320 Kb	360 Kb	720 Kb	1.2 Mb
HC disk	160 Kb	180 Kb	320 Kb	360 Kb	720 Kb	1.2 Mb

2.2: Displaying the directory - DIR

If we want to check whether something has been stored on a disk, and if so, whether the system can decipher the directory, we may use the DIR command, for Directory.

Suppose we want to have the directory of the system disk in drive A displayed. Assuming that drive A is the active drive (see page 17), we then type the following command:

```
A>dir <RET>
```

As a result, a summary like the following appears on the screen:

```
 Volume in drive A has no label
 Directory of  A:\

ANSI     SYS      1709   9-16-87  12:00p
APPEND   EXE      5794   9-16-87  12:00p
ASSIGN   COM      1530   9-16-87  12:00p
ATTRIB   EXE     10656   9-16-87  12:00p
BACKUP   COM     30048   9-16-87  12:00p
CHKDSK   COM     11923   9-16-87  12:00p
COMMAND  COM     25332   9-16-87  12:00p
COMP     COM      4183   9-16-87  12:00p
DEBUG    COM     16000   9-16-87  12:00p
DISKCOMP COM      5848   9-16-87  12:00p
DISKCOPY COM      6264   9-16-87  12:00p
EDLIN    COM      7495   9-16-87  12:00p
FIND     EXE      6403   9-16-87  12:00p
FORMAT   COM     13643   9-16-87  12:00p
GRAFTABL COM      6208   9-16-87  12:00p
GRAPHICS COM      7576   9-16-87  12:00p
JOIN     EXE      9612   9-16-87  12:00p
LABEL    COM      2346   9-16-87  12:00p
MODE     COM     15159   9-16-87  12:00p
MORE     COM       282   9-16-87  12:00p
PRINT    COM      8995   9-16-87  12:00p
RECOVER  COM      5369   9-16-87  12:00p
REPLACE  EXE     13886   9-16-87  12:00p
RESTORE  COM     35720   9-16-87  12:00p
SHARE    EXE      8664   9-16-87  12:00p
SORT     EXE      1946   9-16-87  12:00p
SUBST    EXE     10552   9-16-87  12:00p
TREE     COM      3540   9-16-87  12:00p
XCOPY    EXE     11216   9-16-87  12:00p
       29 File(s)     47104 bytes free

A>_
```

In the same way we can display the directory of a disk in drive B:

A>dir b: <RET>

If we want to see the contents of a hard disk, and assuming that C is the indication of the drive in question, we type:

A>dir c: <RET>

We act in the same way if, for instance, drive B is the active drive instead of drive A. If we want to display the directory of the disk in drive A in such a situation, we type:

B>dir a: <RET>

If it concerns the directory of the disk in drive C, the command will be:

B>dir c: <RET>

As we can see on page 28, the displayed directory of a floppy disk or hard disk is made up of five columns.

The first column contains all kinds of mysterious names such as ANSI, ASSIGN, and ATTRIB. These are file names.

The second column shows terms such as COM, EXE and SYS, which are the extensions belonging to the file names. In the next chapter, on page 48 and page 53, we will explain what files are and what is the meaning and purpose of these extensions.

In the third column we find the size of each file, expressed in the number of bytes (see page 32).

The fourth column indicates the date on which the file was created. If no date was entered after the loading proceedings, a standard date is included in this column, for instance 1-01-80.

The fifth column shows the time at which the file in question was created. An 'a' following the time indicates AM, meaning morning, and a 'p' PM, for afternoon. 12:01a therefore means one minute past midnight.

The end of the directory is marked by a specification of the number of files on the disk in question and the available storage, in this case 72528 bytes. What this means will be explained on page 32.

If there are no files on the disk, or rather, if the directory is empty, we receive the following message:

```
File not found
```

If there is no disk in the drive indicated, or if the disk in question cannot be read by the computer, for instance because the system cannot decipher the contents or because the disk in question has not been formatted yet (see page 38), we will see the following error message:

```
Disk error reading drive B
Abort, Retry, Fail? _
```

Then we have, as we will also see on page 185, to type an 'r' if we have forgotten to insert the disk in the drive, or an 'a' if we want the prompt to appear on the screen again.

If there are so many files on the disk that the directory would fill more than one screen, we could add an extra condition to the DIR command in order to gain a clear view.

In the first place, we could add a slash and a 'p' (for page):

```
A>dir/p <RET>
```

In computerese, such an optional addition is called a parameter.

As a result of adding this parameter, the display will stop as soon as the entire screen is filled:

```
 Volume in drive A has no label
 Directory of  A:\

ANSI     SYS      1709   9-16-87  12:00p
APPEND   EXE      5794   9-16-87  12:00p
ASSIGN   COM      1530   9-16-87  12:00p
ATTRIB   EXE     10656   9-16-87  12:00p
BACKUP   COM     30048   9-16-87  12:00p
CHKDSK   COM     11923   9-16-87  12:00p
COMMAND  COM     25332   9-16-87  12:00p
COMP     COM      4183   9-16-87  12:00p
DEBUG    COM     16000   9-16-87  12:00p
DISKCOMP COM      5848   9-16-87  12:00p
DISKCOPY COM      6264   9-16-87  12:00p
EDLIN    COM      7495   9-16-87  12:00p
FIND     EXE      6403   9-16-87  12:00p
Strike a key when ready . . . _
```

After striking a key the next series of names are displayed.

Another way to slow down the display of names is to add the parameter /w (for wide). Thus:

```
A>dir/w <RET>
```

The result is that the names of the files are divided into five columns. The rest of the information such as the date, time and size is not displayed, as appears from the following example:

```
 Volume in drive A has no label
 Directory of  A:\

ANSI     SYS    APPEND   COM    ASSIGN   COM    ATTRIB   EXE
BACKUP   COM    CHKDSK   COM    COMMAND  COM    COMP     COM
DEBUG    COM    DISKCOMP COM    DISKCOPY COM    EDLIN    COM
FIND     EXE    FORMAT   COM    GRAFTABL COM    GRAPHICS COM
JOIN     EXE    LABEL    COM    MODE     COM    MORE     COM
PRINT    COM    RECOVER  COM    REPLACE  EXE    RESTORE  COM
SHARE    EXE    SORT     EXE    SUBST    EXE    TREE     COM
XCOPY    EXE
       29 File(s)     47104 bytes free
```

If we are only interested in the files on the disk in drive A beginning with the letter R, we type the following command:

```
A>dir r*.* <RET>
```

As a result, we see the following list:

```
RECOVER  COM     5369   9-16-87  12:00p
REPLACE  EXE    13886   9-16-87  12:00p
RESTORE  COM    35720   9-16-87  12:00p
        3 File(s)     47104 bytes free
```

The command to display the files that have a name of four or less letters, regardless of the extension:

```
A>dir ????.* <RET>
```

In this case the result is:

```
ANSI     SYS     1709   9-16-87  12:00p
COMP     COM     4183   9-16-87  12:00p
FIND     EXE     6403   9-16-87  12:00p
JOIN     EXE     9612   9-16-87  12:00p
MODE     COM    15159   9-16-87  12:00p
MORE     COM      282   9-16-87  12:00p
SORT     EXE     1946   9-16-87  12:00p
TREE     COM     3540   9-16-87  12:00p
        8 File(s)     47104 bytes free
```

In order to display the files without an extension, if any, we type:

```
A>dir *. <RET>
```

In this case nothing happens, for we see:

```
File not found
```

Note that all examples of commands include a full stop. This is essential even though it is not included in the listing of the directory. The full stop is the link between the actual file name and its extension.

Also note that we used two special characters in the previous commands, namely the asterisk (*) and the question mark.

In computerese, such characters used in this way are known as wildcards. This term originates from card games in which certain cards, such as the joker, represent a changing or undefined value.

In these examples the asterisk and the question mark are used as such wildcards. The asterisk symbolises a string or series of numbers and letters of an undetermined length. The question mark replaces a single, undefined character.

When we use a command relating to various files at the same time, we may, as is shown in the above examples, replace the common part of the file names by wildcards. The system automatically finds the file names to which the formula relates and the ones that do not match.

2.3: Checking (1) - CHKDSK

We need not concern ourselves with the technology, but we have to know that a computer distinguishes the values 0 and 1 for all data and instructions. These two values are known as bits - an abbreviation of binary digits.

The letters of the alphabet, numerals, punctuation marks and instructions are all converted into a string of eight bits, or to be more exact, into seven bits, ordered in accordance with the system prescribed by the ASCII table (see page 68), plus an extra bit for verification purposes. Such a string of eight bits is called a byte. The size of the computer's memory and the storage capacity of a floppy disk and a hard disk equals the number of bytes (or characters) that can be stored maximally. Memory and storage capacity are measured in units named K or Kb (for kilobyte, or 1024 bytes) and M or Mb or Meg (for megabyte, or over a million bytes).

An A4-sized sheet of paper can contain about 60 lines of an average of 75 characters, including spaces, that is, 4500 characters in total. In other words, 4500 bytes, or over 4 Kb.

By means of the CHKDSK command (for Check Disk) we can obtain an on-screen survey of the available storage capacity and the way it is divided. For instance, if we want to inspect the capacity of the disk in drive A, we type the following command:

```
A>chkdsk <RET>
```

If we want to know what is on the disk in drive B, the command is:

```
A>chkdsk b: <RET>
```

In the case of a disk of a standard size, the following may be the result:

```
362496 bytes total disk space
328704 bytes in 12 user files
 33792 bytes available on disk

655360 bytes total memory
618496 bytes free
```

The first line indicates the total capacity of the disk in question: in this case 362496 bytes, or 360 Kb. This number can be larger or smaller, dependent on the type of the drive we are using (see page 27) and the way in which the disk has been formatted (see page 38).

The second line of the survey reveals the number of files stored (see page 47) and their total capacity, and the third line indicates the available storage capacity on the disk in question.

The penultimate line shows the number of bytes of the total memory. This number concerns the RAM, that is, the computer's own working memory (see page 25). In this example it amounts to 640 Kb.

From the last line we may understand that there are still 618496 bytes available in the working memory for us to use.

The heading of the survey may sometimes look like this:

```
Volume TEST DISK  created Dec 19 1988 1:18p

   362496 bytes total disk space
        0 bytes in 1 hidden files
   362496 bytes available on disk
```

This means that the disk was given a label on the date stated, in this example TEST DISK. On page 42 we will see how this is done.

Note that the survey includes the message 0 bytes in 1 hidden files. This indicates the presence of the label on the floppy disk or hard disk (see page 42).

It is possible that the survey contains a different message with regard to the hidden files, for instance:

```
38912 bytes in 2 hidden files
```

In this case the hidden files take up 38912 bytes.

The two files have been stored on the floppy disk or hard disk, but have not been included in the directory (see page 28). This usually concerns system files, that is, the files containing the part of the system needed for loading. We will further discuss this subject on page 48.

If we want to know the names of the hidden files and whether they are indeed the system files, we add the parameter /v (for verbose) to the CHKDSK command. Thus:

A>chkdsk/v <RET>

If it concerns the disk in drive B, we type:

A>chkdsk b:/v <RET>

In both cases the result is that a series of file names is displayed on the screen. For instance:

```
A:\IBMBIO.COM
A:\IBMDOS.COM
A:\ANSI.SYS
A:\ASSIGN.COM
A:\ATTRIB.EXE
A:\BACKUP.COM
A:\CHKDSK.COM
A:\COMMAND.COM
```

And so on.

The first two names are the hidden files: in this example IBMBIO.COM and IBMDOS.COM. The names could also have been IO.SYS and MSDOS.SYS.

If the floppy disk (or hard disk) contains so-called bad sectors (see page 41), we would receive another additional message, for instance:

```
5120 bytes in bad sectors
```

2.4: Duplicating (1) - DISKCOPY

It is advisable to make a copy of the entire system disk first and to store the original in a safe place. If anything happens to the copy, we still have the original disk.

Our working method is as follows. We put the system disk in drive A and type the following formula exactly as it is shown here:

```
A>diskcopy a: b: <RET>
```

If we make a typing error, we can correct it by means of the methods we learned on page 18, either the third or the fourth method.

Note that we first typed a: after the term DISKCOPY in order to indicate the source drive, and then b: to indicate the target drive. The result is:

```
Insert source diskette in drive A:

Insert target diskette in drive B:

Strike any key when ready
```

The disk to be copied should now be inserted in drive A. In this case we want to copy the system disk itself, so there is no need to change it.

Next, we put a blank disk in drive B. If we have done this, we strike any key to indicate that the procedure may start.

Whatever is shown on the screen depends on the type of drive. Some old types only are single sided, which results in only half of the disk's capacity being used (see page 27). We then see:

```
Copying 1 side(s)
```

But if the drive, like almost all drives nowadays, can fill double-sided disks, the message is:

```
Copying 2 side(s)
```

If the target disk has not been formatted yet (see page 38), we also see:

```
Formatting while copying
```

A few seconds later, we should see the following message, showing that the command has been carried out properly:

```
Copy complete
```

The system then asks:

```
Copy another (Y/N)?_
```

If we want the system to make a second copy, we type a 'y', for yes:

```
Copy another (Y/N)?y <RET>
```

As a result, we again see:

```
Insert source diskette in drive A:

Insert target diskette in drive B:

Strike any key when ready
```

If we have finished, in other words, if we no longer need the DISKCOPY procedure, we type an 'n', for no. Thus:

```
Copy complete

Copy another (Y/N)?n <RET>

A>_
```

Drive B now contains an exact copy of the disk in drive A. We remove the original from drive A and store it in a safe place.

DISKCOPY also works the other way around, that is, from drive B to drive A. To demonstrate, we leave the copy of the system disk we have just made in drive B. We insert a blank disk in drive A and this time we first type:

```
A>b: <RET>

B>_
```

Now drive B is the active drive. Next, we type:

```
B>diskcopy b: a: <RET>
```

If we want to maintain drive A as the active drive, we may combine these commands in the following way:

```
A>b:diskcopy b: a: <RET>
```

Note that the command is preceded by a 'b' and a colon. In this way we inform the computer that drive A must remain the active drive, even though the code of the DISKCOPY program is on the disk in drive B.

Consequently, the instructions have now been adapted:

```
Insert source diskette in drive B:

Insert target diskette in drive A:

Strike any key when ready
```

The procedure is otherwise the same.

It may be necessary to use a DISKCOPY procedure to copy a single-sided disk in a double-sided drive (see page 27). For instance, because the copy is needed for a computer with single-sided drives (see page 27). In that case, we have to warn the operating system in advance by adding the parameter /1 to the command. Thus:

```
A>diskcopy a: b:/1 <RET>
```

The procedure is otherwise the same as on the previous pages. If there is only one disk drive, for instance if the space of a B drive is taken up by a hard disk, the DISKCOPY procedure is slightly different.

First, we insert the disk to be copied in the available drive (by definition, drive A), and next we type the following command. Thus:

```
A>diskcopy <RET>
```

Or, if we want to copy a single-sided disk in a drive meant for double-sided disks:

```
A>diskcopy/1 <RET>
```

In both cases the system's answer is:

```
Insert source diskette in drive A:

Strike any key when ready
```

If we then strike a key, as requested, we see after a few seconds, the usual messages. And as soon as the contents of the disk have been adopted in the working memory, we will see:

```
Insert target diskette in drive A:

Strike any key when ready
```

We then remove the disk from the drive and replace it by a blank disk, and strike a key. The result is:

```
Insert source diskette in drive A:

Strike any key when ready
```

Subsequently, we put the original disk in the drive again and repeat this exchange a number of times until we receive the message that the procedure has been completed:

```
Copy complete

Copy another (Y/N)?_
```

Another method to copy all or part of the contents of a disk is the COPY *.* command. See page 55.

Since the introduction of the system's 3.2 and 3.3 versions, we can also use a command called XCOPY. We will review it on page 178.

2.5: Comparing (1) - DISKCOMP

Almost all versions of MS-DOS amd PC-DOS provide us with the possibility to check whether a DISKCOPY procedure has been accomplished without any errors. In other words, whether each byte of the copy corresponds with the original. In order to find this out, we type:

```
A>diskcomp a: b: <RET>
```

In both cases we see the following instructions:

```
Insert first diskette in drive A:

Insert second diskette in drive B:

Strike any key when ready
```

Now we have to insert one of the disks to be compared in drive A, and the other disk in drive B before striking any key.

As soon as this procedure has been completed, which may take a while, we may receive a message like:

```
Diskettes compare ok

Compare more diskettes (Y/N)?_
```

By typing a 'y' or an 'n', we can indicate whether or not we want to compare another disk.

If the procedure results in having found a difference between the two disks, we may expect a message like:

```
Compare error(s) on
Track 04, Side 0
```

If such messages follow each other rapidly, it is obvious that we are dealing with two disks of different contents. In that case it is advisable to break off the procedure by means of the ^C or ^<BREAK> control instruction.

If we only have one drive, the command is:

```
A>diskcomp <RET>

Insert first diskette in drive A:

Strike any key when ready
```

Then we insert the first disk in the drive and strike any key. What happens is largely the same as in the DISKCOPY procedure (see the previous page).

2.6: Formatting - FORMAT

Before we can store data on a disk, the disk has to be equipped with a kind of electronic map or blueprint first. The computer then knows how the drive has to operate. Such a procedure is called formatting, or initialising the disk. The command to bring about such a procedure is FORMAT.

In the course of our using the system, we will have to use it rather often; in any case each time we want to use a new disk, or if we have to clear or repair a disk that has been used before.

However, we have to be very cautious with the FORMAT command. The computer does not distinguish a used disk from an empty one. If we make an error in the formula of the command, the entire contents of a disk may be removed irrevocably.

Let's perform an experiment. We insert a copy of the system disk in drive A and a blank disk in drive B. Next, we type:

```
A>format b: <RET>
```

The result of this command is that the FORMAT program is stored in the computer's memory, i.e. is loaded, which is confirmed by the following message:

```
Insert new diskette for drive B:
and strike ENTER when ready
```

For reasons of safety, we remove the copy of the system disk from drive A and strike the ENTER key, i.e. the RETURN key.

We receive the following confirmation that the procedure is in operation:

```
Formatting...
```

At the same time we usually hear a strange ticking noise from the drive. The screen may also show the term 'head' or 'cylinder' or both, invariably followed by a number rapidly increasing.

As soon as the formatting procedure has been completed, we are to receive the following confirmation:

```
Formatting...Format Complete
```

This is followed by a calculation of the total and still-available storage capacity of the disk. If we are working with a normal double-sided drive and a DSDD disk, we see:

```
362496 bytes total disk space
362496 bytes available on disk
```

Then the system will ask whether we want to format another disk:

```
Format another (Y/N)?_
```

Now we can indicate whether we want to format a second disk in the same drive. If so, we type a 'y' (for yes). As a result, we are again asked to insert a disk in drive B. Striking the RETURN key will start the procedure:

```
Insert new diskette for drive B:
and strike ENTER when ready
```

On the other hand, if we type an 'n' (for no), the prompt appears on the screen again. Thus:

```
Format another (Y/N)?n

A>_
```

If we only have one disk drive, for instance if the computer is equipped with a hard disk, we have to restrict the procedure to the available drive. By definition, this is drive A. In that case the command has to be:

```
A>format <RET>

Insert new diskette for drive A:
and strike ENTER when ready
```

After this instruction on the screen, we have to replace the system disk by the blank disk. If we forget to change the disks, or if we accidentally strike the RETURN key, the system disk will be formatted so that the contents are erased irrevocably. This is the reason why we should always work with a copy of the system disk instead of the original.

For the rest, the procedure results in the same series of messages as on the previous page, for instance:

```
Formatting...Format Complete
Format another (Y/N)?_
```

Just like a floppy disk, a hard disk has to be formatted - as we will see on page 81. Normally, the supplier of the computer has taken care of this. If not, and a DIR command will prove this (see page 28), we have to do it ourselves. In that case we type:

```
A>format c: <RET>
```

This results in a message like:

```
Press ENTER to begin formatting drive C.

_
```

Should we ever receive such a message on the screen when this is certainly not our intention, we immediately have to give a ^<BREAK> or ^C instruction (see page 71) in order to prevent the FORMAT procedure from being started by accident. Fortunately, most recent versions of the system give us a warning like the following:

```
WARNING, ALL DATA ON NON-REMOVABLE DISK WILL BE LOST!
Proceed with Format (Y/N)? _
```

Sometimes it is necessary to format a disk for less than the maximum storage capacity, for instance 8 sectors per track, in a standard, double-sided type of drive, in order to guarantee the usability under any version of the system, including the 1-versions.

Assuming that we have a B drive at our disposal, the formula is:

```
A>format b:/8 <RET>
```

If we want to format the disk for 180 Kb, for instance because it has to be used in a different computer with a single-sided drive, we have to add the parameter /1 to the FORMAT command. Thus:

```
A>format b:/1 <RET>
```

We may also combine the previous parameters so that we restrict the storage capacity of the disk to one single side, with eight sectors per track:

```
A>format b:/1/8 <RET>
```

As usual, we see:

```
Insert new diskette for drive B:
and strike ENTER when ready
```

At the end of the procedure, we see the following statement:

```
   160256 bytes total disk space
   160256 bytes available on disk

Format another (Y/N)?_
```

If we work on a computer with a High Capacity drive (see page 26), for instance an IBM PC AT, or a similar type, the FORMAT procedure will be carried out in the usual way. Thus:

```
A>format <RET>
```

The calculation and capacity at the end of the procedure look like this:

```
  1213952 bytes total disk space
  1213952 bytes available on disk
```

A parameter only available in the 3-versions of the system is /4. If we work on a computer with a QD or HC drive (see page 26), we can arrange that a normal disk in this drive is adapted for a storage capacity of 360 Kb, instead of 1.2 Mb, by adding the aforementioned parameter to the command:

```
A>format/4 <RET>
```

This possibility may come in handy if we do not have an HC disk at our disposal.

However, we have to remember that a disk formatted by means of this command may not always be read reliably in a standard drive.

A formatting procedure is usually the only way out when we are confronted with a damaged floppy or hard disk that can no longer be read by the computer. Should something like that be the case, then this will sooner or later be shown in some kind of error message (see page 183 and subsequently).

We first have to make a copy of the disk in question by means of a DISK-COPY procedure (see page 34) since a FORMAT procedure would result in erasing the contents from a floppy disk or hard disk irrevocably. After this, we store the copy in a safe place.

Next we start a FORMAT procedure in the usual way. Thus:

```
A>format b: <RET>
```

If we only have one disk drive, for instance because the computer has a hard disk, we restrict the procedure - as always - to the available drive. Thus:

```
A>format <RET>
```

In both cases the rest of the procedure is carried out in a similar way:

```
Insert new diskette for drive A:
and strike ENTER when ready
```

During the formatting procedure, the damaged parts or bad sectors are fenced off, so to speak. Anything we store on the disk in a later stage will be carefully routed around the fence.

After completion of the procedure, we see a message like the following:

```
   362496 bytes total disk space
     4096 bytes in bad sectors
   358400 bytes available on disk

Format another (Y/N)?_
```

Note that the message '4096 bytes in bad sectors' is added to the survey.

This means that we can use the disk in a normal way, though with the available storage capacity reduced by the number of bytes stated. The amount of space finally available is stated in the last line.

The survey produced by a CHKDSK command (see page 32) will give us a confirmation like the following:

```
A>chkdsk b: <RET>

   362496 bytes total disk space
     4096 bytes in bad sectors
   358400 bytes available on disk

   655360 bytes total memory
   618496 bytes free

A>_
```

On the repaired disk we can now put the contents of the copy we made at the beginning of this procedure and stored safely. To do so we have to start a COPY *.* procedure (see page 55).

2.7: Entering and changing a label

The FORMAT procedure enables us to give a disk a so-called volume label. This can be a description of the contents of the disk in question, a name, a code or anything else. The system automatically adds the date and time.

If we want to give a disk such a label, we have to inform the system of our intention in advance. We do so by adding the parameter /v (for volume) to the formula of the FORMAT command. Thus:

```
A>format b:/v <RET>
```

The procedure is initially identical to that on the previous pages. But then the following message appears on the screen:

```
Volume label (11 characters, ENTER for none)? _
```

We should, however, realise that (in the system's 2-versions) we can only change or remove the label by reformatting the disk. Therefore, we should think carefully about the formula of the label.

Another handicap is that the label may not contain more than 11 characters, numerals or punctuation marks, while the same rules apply as in naming files (see page 53).

If we decide not to add a label after all, we just hit the RETURN key.

Suppose we want to give our disk the label TEST DISK. As soon as we have passed it on to the system by hitting the RETURN key, the prompt appears on the screen again - without further statements. If we then give a DIR command, the result is as follows:

```
 Volume in drive B is TEST DISK
 Directory of B:\

File not found

A>_
```

If we give a CHKDSK command instead of a DIR command (see page 32), the screen will show us the label as well as the date and time when it was created. For instance:

```
A>chkdsk b: <RET>

 Volume TEST DISK created Dec 17 1988 16:39

   362496 bytes total disk space
        0 bytes in 1 hidden files
   362496 bytes available on disk

   655360 bytes total memory
   618496 bytes free
```

Note that the survey contains the statement '0 bytes in 1 hidden files'.
This message shows that the label is included as an entry in the directory, with the same status ('hidden') as the system files (see page 48). However, it has no meaning at all - hence the size of 0 bytes.

If we do not want to display the directory, we can type the command VOL (see page 19) instead of a DIR or CHKDSK command. Thus:

```
A>vol b: <RET>
```

The reply will then be restricted to the following:

```
Volume in drive B is TEST DISK
```

If the disk has not been given a label, we receive the following message:

```
Volume in drive B has no label

A>_
```

In the 2-versions of the system, we could create a label by means of the FORMAT procedure in the way described on the previous page, but we could not change or erase it just like that.
This has changed with the introduction of the 3-versions of MS-DOS and PC-DOS. From now on, the label of any disk, regardless of its type, can be added, changed or erased again at any time.
To demonstrate, we insert a formatted disk in drive B. In order to give it the label VARIANT, for instance, we type the following command, assuming that the system disk is in drive A:

```
A>label b:variant <RET>

A>_
```

Now we can check by means of a VOL or CHKDSK instruction whether the label has indeed been created:

```
Volume VARIANT  created Dec 19 1988 1:39p
```

If we want to erase the label from a disk in drive B, we type:

```
A>label b: <RET>
```

The following answer appears on the screen:

```
Volume label (11 characters, ENTER for none)? _
```

We only have to strike the RETURN key again so that the prompt appears on the screen again. If we then, by way of trial, give the VOL command, the label turns out to have been erased:

```
A>vol b: <RET>

Volume in drive B has no label
```

2.8: Transferring the system

The previous pages explained the meaning of the FORMAT procedure. One facility has been left aside, namely to provide a disk with the system files at the same time as it is formatted. In other words, to provide it with the two 'hidden files' and the COMMAND.COM file.

The parameter /s is used with the FORMAT command to format a disk and to transfer the system on to it at the same time. Assuming that the system disk is in drive A and the disk to be formatted in drive B, we type:

```
A>format b:/s <RET>
```

At first the procedure is otherwise the same, for we see:

```
Insert new diskette for drive B:
and strike ENTER when ready
```

Even after we strike the RETURN key, the procedure is the same as usual:

```
Formatting...Format Complete
```

But then this message appears:

```
System transferred
```

And next, a survey like the following:

```
362496 bytes total disk space
 60416 bytes used by system
302080 bytes available on disk

Format another (Y/N)?_
```

The electronic blueprint on the disk now has a different form - a number of tracks and sectors have been reserved for (and subsequently taken up by) the system files: 60416 bytes in total, as the survey shows. From the survey it also appears that this space has been taken from the available storage capacity.

We may now carry out an experiment. We replace the system disk in drive A by the disk we just formatted. Next, we carry out a RESET procedure (see page 18). It turns out that the system can be loaded without any problem.

If we then give a DIR command, we see:

```
A>dir <RET>

  Volume in drive A has no label
  Directory of  A:\

COMMAND  COM     25332   9-17-87  12:00p
        1 File(s)    302080 bytes free
```

If we only have one drive, for instance if we have an AT-type computer with a hard disk, the procedure will be carried out as follows. Assuming that the system disk is in drive A and the disk to be formatted in drive B, we type:

```
A>format/s <RET>
```

As expected, the system's answer is:

```
Insert new diskette for drive A:
and strike ENTER when ready
```

Next, we replace the system disk by the blank disk to be formatted and strike the RETURN key. The procedure will then also be the same as usual, finally resulting in this message:

```
System transferred

  1228800 bytes total disk space
    60416 bytes used by system
  1168384 bytes available on disk
```

If desired, we may combine the parameter /s with /v (see page 42). As a result, the disk is given a label at the same time. Thus:

```
A>format b:/s/v <RET>
```

Or, if we only have one drive:

```
A>format/s/v <RET>
```

In both cases we receive the usual messages:

```
Formatting...Format Complete
System transferred

Volume label (11 characters, ENTER for none)? _
```

It may sometimes be necessary to transfer only the system files to a disk, for instance to a disk with a ready-to-use program in which the designer or manufacturer has not included system files for reasons of copyright, but who has reserved the necessary space.

In that case, a FORMAT procedure with the parameter /s cannot be used, since it would erase the contents of the disk entirely. In such a situation (but only then) we have to give a different command. Assuming that our system disk is in drive A and the disk to be transferred in drive B, we type:

```
A>sys b: <RET>
```

The only on-screen result is:

```
System transferred
```

However, the disk does not contain the indispensable COMMAND.COM file yet. To get that on the disk as well, we have to COPY it (see page 55).

3: FILES

In the previous chapter we saw how to prepare a floppy disk or hard disk for storing data and how to check whether this has been done, and if so, how much space is still available. In addition, we have learned the meaning of bits and bytes, kilobytes and megabytes. But how can the computer make sense out of the thousands of bytes on all those sides, tracks and sectors?

The answer is that this is precisely the main task of the operating system. It uses a list of files to keep track of what has been stored, where and when.

A file is a collection of one or more bytes, with a minimum of one and a maximum of 32 Mb, that is, more than 32 million bytes.

The actual contents of a file are determined by the number, the order and the meaning of the stored bytes. These contents may be anything, like a normal text we can read, varying from a single character to, for instance, the contents of this manual. In that case we are dealing with a text file.

The contents of a file may also consist of a large or small program. In other words, a series of instructions, defined in accordance with the fundamentals of a so-called computer language like Pascal or BASIC (see page 155). If that is the case, we refer to this file as the program file.

Finally, a file may consist of data stored in a certain pattern by means of a database management program like PC-File, or dBASE. In that case we refer to it as a data file.

We have created a file (of any type) as soon as we have thought of a name and have passed it on to the operating system. The number and type of characters used in the file name is somewhat restricted (see page 53), but in principle we are free to choose.

After receiving the name, the system includes it in the directory (see page 28), and automatically adds the date and time of creation, as well as a reference to the so-called File Allocation Table, known to programmers as the FAT. This is, so to speak, the map or electronic blueprint we discussed in the previous chapter. In this map, the system precisely registers where each file has been stored on the floppy disk or hard disk. In other words, on which sides, tracks and sectors the series of bytes are.

Then the system locates the first empty track and sector on the disk indicated and reserves this space for the file to be stored by making a note in the FAT.

As soon as the storing procedure is completed, the system finishes the bookkeeping by determining the size in bytes of the file that we have just created.

When the floppy disk or hard disk is equipped with the electronic map, we can display its contents by giving a CHKDSK command (see page 32) or display part or the entire directory by giving a DIR command (see page 28).

In this chapter we will learn how to create a file ourselves and how, if desired, to change, reproduce, copy, print, or erase a file. In other words, how to manipulate a file. In addition, we will learn what to do when a file, or even the entire directory of a disk, turns out to be mixed up. To be more precise, when the FAT's bookkeeping is a mess.

3.1: Utilities and system files

Let's take another look at the directory of our system disk. We see that the list contains a rather varied collection. Most file names appear to have the extension .COM or .EXE, but there is a number of file names ending with .SYS as well. In most cases (though not in all), the manufacturer of the computer has been kind enough to provide the main files on the system disk with the appropriate date and time.

In any case, this collection of files forms the operating system. In fact, we should add, in a wider sense; for we do not need most of these files to start the computer and to carry out a large number of instructions.

To demonstrate, we carry out the following experiment. We erase all files from a copy of the system disk, apart from COMMAND.COM. How to do this will be explained on page 61. Next, we insert this reduced system disk in drive A and start a RESET procedure (see page 18). As a result, the computer acts as usual, as if there is nothing wrong. We even can use the keyboard and give commands such as CLS, DATE, DIR, TIME, VER and VOL as if nothing has happened. On the other hand, if we give commands such as CHKDSK, DISKCOMP, DISKCOPY, FORMAT or SYS, the system does not recognise them.

The reason for this derives from the design of the operating system. We have to imagine that the operating system consists of a centre surrounded by a certain number of sections. These sections are formed by the files that we just erased from the disk, such as ANSI.SYS, ASSIGN.COM, ATTRIB.EXE, BACKUP.COM etc. These files appear not to be essential to the computer's functioning, for the system is loaded anyway. They do, however, perform a certain supplementary or special task. Hence the indication utilities. We might also call them utility programs. Dependent on the operations we have to carry out, we will use certain utilities more often than others.

In order to use a utility like CHKDSK, the file CHKDSK.COM has to be on the system disk. We can find this out by means of a fairly simple method. We only have to transfer a copy of CHKDSK.COM to our reduced disk. As a result, the CHKDSK command will be in operation again.

In a similar way we can ascertain that it is essential that the files FORMAT.COM or CHKDSK.COM are on the disk in order to carry out the FORMAT or CHKDSK command. Since these utilities operate more or less independently of the actual operating system, we can store them on disks other than the system disk and activate them.

As stated above, the directory of the system disk sometimes indicates that the utilities have been given the same date and time. In this way we can see that the files in question are of the same sort. Or rather, that they are part of the same version of the system. It is not advisable to use utilities of different versions of the system indiscriminately, since this involves the risk of a breakdown.

The operating system in the narrower sense of the term, i.e. the centre of the system, that is, without the utilities, consists of three files. Only COMMAND. COM is viewed in the directory. As a result of a technical little trick in the FAT (see page 47), the other two files remain hidden in DIR commands. This explains the term hidden files we saw on page 33.

COMMAND.COM contains, among other things, the instructions to process everything we pass on to the computer through the keyboard, and the program code for the commands DATE and TIME we saw in the previous chapter, as well as the commands COPY, COPY CON:, DEL, DIR, ERASE and TYPE in this chapter.
Loading the system largely means that both the hidden files and the COMMAND.COM file are stored in the computer's memory. This is why the aforementioned commands, unlike the commands to operate the utilities, can be carried out without the system disk.
There is the possibility of displaying the two hidden files, as we saw on page 33. For this purpose, we have to give a CHKDSK command, followed by the parameter /v (for verbose). Thus:

```
A>chkdsk/v <RET>
```

Assuming that the system disk is in drive A, the result is:

```
Directory A:\
      A:\IO.SYS
      A:\MSDOS.SYS
      A:\COMMAND.COM
```

This is followed by the file names of the utilities, if any, plus the data we saw on page 32. For instance:

```
362496 bytes total disk space
 38912 bytes in 2 hidden files
188416 bytes in 31 user files
135168 bytes available on disk

655360 bytes total memory
618496 bytes free
```

In this example we see that the hidden files are named IO.SYS and MSDOS. SYS.

If we work with a recent or an IBM version of the operating system, we receive a message like the following:

```
Directory A:\
      A:\IBMBIO.COM
      A:\IBMDOS.COM
      A:\ANSI.SYS
      A:\ASSIGN.COM
      A:\ATTRIB.EXE
      A:\BACKUP.COM
      A:\BASIC.COM
      A:\BASICA.COM
      A:\CHKDSK.COM
      A:\COMMAND.COM
      A:\COMP.COM
      A:\DISKCOPY.COM
```

And so on.
Conclusion: the hidden files are called IBMBIO.COM and IBMDOS.COM.

3.2: Creating a text file - COPY CON:

There are several ways in which we can create a file ourselves. Mostly, we will use the utility EDLIN to do so (see page 95), or a word-processing program, but there is an alternative, fast method to create draft and other small files.

Let's use this method to create a file named NOTES. We do not need an extension. We type the following command:

```
A>copy con: notes <RET>
```

If we want the file to be on the disk in drive B, we type:

```
A>copy con: b:notes <RET>
```

As we can see on the screen, the cursor has moved to the beginning of the next line, ready to process whatever we type. We can use the keyboard more or less like a typewriter.

At the end of each line, text or no text, we have to strike the RETURN key. The cursor will then be placed at the beginning of the next line.

We can correct typing errors by means of the backspace key or the ^H instruction (see page 18). If we want to delete a whole line of text, we can hit the ESC key (see page 18). If we want to stop the procedure for any reason, we have to type the ^C or ^<BREAK> instruction (see page 71).

As soon as we have finished, we give the ^Z instruction or strike the F6 key, in both cases followed by tapping the RETURN key.

After having completed the file, we should see the following text on the screen, provided that this is what we have typed:

```
A>copy con: b:notes
It is advisable to enter the date and time when
loading the system, so that we later can check
when a file like this was created.
^Z
         1 File(s) copied

A>_
```

A DIR command shows that the text has indeed reached the disk:

```
A>dir b: <RET>

 Volume in drive B has no label
 Directory of  B:\

 NOTES              134  12-17-88  10:17a
        1 File(s)    361472 bytes free

A>_
```

3.3: Reproducing (1) - TYPE

There are several ways to reproduce the contents of a file on the screen. One of them is the TYPE procedure.

For instance, if we want to see the NOTES file we created on the previous page, we use the TYPE command as follows:

A>type b:notes <RET>

Immediately the contents of the file start passing the screen:

```
It is advisable to enter the date and time when
loading the system, so that we later can check
when a file like this was created.
```

As soon as the end of the file has been reached, the prompt will appear on the screen again:

A>_

Using this method, we can reproduce the contents of every type of file, but only in the case of a text file are we certain that we will see a legible text, as a few experiments will prove.

If desired, for instance, in order to study a specific part of the file, we can interrupt a TYPE procedure. To do so, we use the control instruction ^<NUM LOCK> (see page 71), which results in the flow of text stopping.

After striking any key, the flow will start again.

If we want to stop a TYPE procedure for any reason, we have to give the control instruction ^C or ^<BREAK> (see page 71).

If the file in question is not found in the indicated drive, we receive the following message:

```
File not found

A>_
```

We can also use a TYPE procedure if we want to print the contents of a file on paper. In that case we add the control instruction ^P or ^<PRT SC> (see page 71) to the actual command, before striking the RETURN key. Thus:

A>type b:notes <^P> <RET>

Assuming that the printer is connected to the computer, it will operate immediately after issuing the command.

In order to break off the contact with the printer, we have to give a ^P or ^<PRT SC> instruction again immediately after printing.

3.4: Renaming - REN

Suppose we are not happy with the name NOTES of our file and prefer to call it TEXT. For this purpose we have to type the following command:

```
A>ren b:notes text <RET>

A>_
```

Note that we first type the old name, then a space and finally the new name. We need not indicate the drive before the new name.

The rules for naming files (see next page) also apply for this procedure.

There is no confirmation that the command has been carried out. In order to check whether the change has been adopted, we have to give a DIR command:

```
A>dir b: <RET>

 Volume in drive B has no label
 Directory of  B:\

TEXT             134  12-17-88  10:17a
        1 File(s)    361472 bytes free

A>_
```

Note that the file's size, date and time entries have not been changed.

Suppose we have written three files, named NOTES.1, NOTES.2 and NOTES.3, and we want to give them all TEXT as a new name, along with the relevant numeral, thus: TEXT.1, TEXT.2 and TEXT.3.

Obviously, we can rename them one by one. But we can also use the wildcard symbols which were introduced on page 31. In that case we only need a single command:

```
A>ren b:notes.? text.? <RET>
```

Should we want to, we can in the same way rename .COM and .EXE files as well, but the extensions themselves may never be changed.

Suppose we want to change the CHKDSK command because we keep making mistakes in typing it and that we prefer to shorten the name to the letter C. In that case we maintain the extension and change the actual name:

```
A>ren chkdsk.com c.com <RET>
```

In this command we may also use the wildcards. As a result, we come up with the following, concise formula of the command:

```
A>ren ch*.* c.* <RET>
```

The restricted space reserved on the disk for the directory is the reason why file names are allowed only to have a limited length. This length is restricted to a maximum of eight characters for the actual file name, plus another three characters for a possible extension.

The name and the extension can be made up of any letters and numerals. We need not make a distinction between capitals and lower case, since the system automatically converts all letters into capitals in the directory.

In addition to letters and numerals, we can also use a number of other characters on the keyboard, namely ! @ # $ % ^ & ( ) - _ { } ' and `. It is not permitted to use the space, full stop, comma, colon, semi-colon and the signs * + [ ] ? | \ < > and /, since these signs have a specific significance for the system.

Extensions are not compulsory, but some have a specific meaning for the computer and it goes without saying that we have to be aware of this.

The most important standard extensions, plus their meanings, are:

.BAK - for Back up file. Only plays a part in editing text files. See page 99.

.BAS - for Basic file. A program file created in the BASIC computer language. See page 155.

.BAT - for Batch file. A program file made up of commands from the operating system. If we intend to use such a .BAT program, we need not type the extension. See page 145.

.C - for C file. A program file created in the C computer language.

.COM - for Command file. A program file made up of computer codes which can be executed at any time. In order to activate a .COM program, we need not type the extension.

.DOC - for Document file. A text file in so-called ASCII characters. See page 68.

.EXE - for Executable command file. Like .COM, a program file made up of machine language which can be executed at any time - hence the term. In order to activate an .EXE program, we need not type the extension.

.OVR - for Overlay file. Usually a part of an application program.

.PAS - for Pascal file. A program file created in the Pascal computer language.

.SYS - for System file, a file containing the system codes. See pages 48 and 126.

.TXT - for Text file. Like .DOC, a text file in ASCII signs. See page 68.

.$$$ - a temporary, damaged or empty file. It is automatically created in certain situations and it usually disappears again without trace. See the example on page 97.

3.5: Reproducing (2) - MORE

A disadvantage of a TYPE procedure (see page 51) is that a file flows on the screen rapidly, and that we have to be very quick in striking two keys (the CTRL key and the NUM LOCK key) in order to interrupt the flow of data.

Instead of TYPE, we may also use the MORE command. However, the MORE.COM file has to be on the disk in the active drive.

Suppose we want to take another look at our TEXT file on the disk in drive B, but in parts. In that case we type the command as follows:

```
A>more <b: text <RET>
```

The result of the command is that the contents of the file appear on the screen in blocks of 23 lines, that is, in the size of the screen. As long as we have not reached the end of the file, each block of lines is followed by:

```
--- More ---
```

Striking any key suffices to have the next series of lines appear on the screen, and so on, until we have seen the whole text, or we break off the procedure through the control instruction ^C or ^<BREAK> (see page 71).

3.6: Verifying - VER

It often happens that something is sent from the computer's memory to a floppy disk or hard disk. An example is the procedure in which we create a text file (see page 50). If we want to be absolutely sure that the created text has been stored on the disk without the slightest error or failure, we have to use a command that verifies such a transaction byte by byte, namely:

```
A>verify on <RET>
```

Should there be a failure or anything else wrong, we receive an error message.

The command to verify remains in operation until we turn off the computer, or start a RESET procedure, or give a command to the contrary. This is:

```
A>verify off <RET>
```

If we do not know whether VERIFY is on or off, we use the bare form of the command to find out. Thus:

```
A>verify <RET>

VERIFY is off
```

3.7: Copying (1) - COPY

The use of the DISKCOPY command (see page 34) has several disadvantages. In the first place, it is rather a slow procedure. In the second place, it does not enable us to copy separate files.

Fortunately, the system is equipped with the COPY program, which overcomes both disadvantages: it works fast and offers us the possibility to make a selection of one or more files that we wish to copy.

Assume the system disk is in drive A, and a blank, formatted disk in drive B. In order to make an exact copy of the files on the system disk to the disk in drive B, we type the following command:

A>copy a:*.* b:/v <RET>

Note that we use the asterisk - one of the wildcard symbols we learned to use on page 31.

Also note that we added the parameter /v (for verify) to the formula. This causes an automatic verification that the copy is identical to the original file. In that case, a VERIFY command (see previous page) is unnecessary.

Since the source drive is the active drive also, as the prompt indicates, we may leave out a: in the formula. Thus:

A>copy *.* b:/v <RET>

A:COMMAND.COM
A:ANSI.SYS
A:ASSIGN.COM

And so on. We can follow the proceedings on the screen as each file is copied. At the end of the procedure, we receive a message concerning the number of files copied:

31 File(s) copied

A>_

The command for the same procedure in the reverse direction is:

A>copy b:*.* a:/v <RET>

Or, because A is known as the active drive:

A>copy b:*.*/v <RET>

If we only have one drive, we can use exactly the same commands.

First, the system considers the drive as drive A, and adopts the files in it, stores them in the memory, then considers the drive as drive B and finally transfers these files to this drive.

The COPY commands we became acquainted with on the previous pages are related to each other. The basic form is the COPY command that results in a duplicate of one file. In order to copy the TEXT file from the disk in drive A to the disk in drive B, for instance, we type the following formula:

```
A>copy b:text a: <RET>
```

This results in a similar answer:

```
                1 File(s) copied

A>_
```

In the actual COPY command, we first state the source, in this case drive B, and second the place of destination, that is, drive A. As we saw on the previous page, we may leave out the reference to the active drive (a:) in the formula of the command.

In addition, we may also add the parameter /v to the formula in this form, in order to check that the copy and the original are identical. Thus:

```
A>copy b:text/v <RET>
                1 File(s) copied

A>_
```

However, we have to be careful with the COPY command. If there is already a file bearing the same name on the disk in the target drive, the command is carried out nevertheless.

Even if the two files differ in size, date and time, the contents of the file are replaced by the contents of the copied file at the place of destination - without any warning in advance.

In such a situation, and if we want to make a copy of a file on the same disk, we have to give a special command so that the copy is given a different name. For example:

```
A>copy b:text a:test/v <RET>
                1 File(s) copied

A>_
```

As we may check by means of a simple method, both the date and time of the copy turn out to be identical to the original. So it is not relevant when the COPY procedure takes place.

If we want to copy a file and add the date and time of the procedure, we add a plus sign to the formula:

```
A>copy b:text a:test+/v <RET>
B:TEXT
                1 File(s) copied

A>_
```

It is also possible to alter only the date and time entries of a file. The command to do so is the following:

```
A>copy b:text+,, b: <RET>
        1 File(s) copied

A>_
```

Note that in this case the plus sign is followed by two commas. This tells the system that we are not dealing with a command to join copies together, in which we also use plus signs (see the following).

Suppose that we want to combine TEXT and NOTES into one file on the disk in drive A and copy it on to the disk in drive B. Assuming that we want to name it DUET, we type the following formula:

```
A>copy text+notes b:duet/v <RET>
```

In this case, we may use a wildcard symbol to abbreviate the command since neither of the file names contains an extension.

```
A>copy *. b:duet/v <RET>
        1 File(s) copied

A>_
```

This command in fact means: copy all the files without an extension on the disk in drive A and join them together to one new file named DUET on drive B and compare the contents with those of the original files.

If we then give a DIR command, we see that a new file named DUET has indeed been added to the disk in drive B:

```
A>dir b: <RET>

 Volume in drive B has no label
 Directory of  B:\

NOTES                  134  12-17-88  10:17a
DUET                   267  12-18-88   2:34p
          2 File(s)  360448 bytes free

A>_
```

Note that the size of DUET is 267 bytes, rather than exactly twice the size of NOTES.

A TYPE command finally shows us the contents of DUET:

```
A>type b:duet <RET>
It is advisable to enter the date and time when
loading the system, so that we later can check
when a file like this was created.
It is advisable to enter the date and time when
loading the system, so that we later can check
when a file like this was created.

A>_
```

COPY procedures are not restricted to the transfer of a file (or a number of files) from one disk to another. The source can also be another part of our computer, for instance the screen and the destination can be a printer, or a modem.

An example of this was given on page 50, when we created our NOTES file by means of a COPY command. In this, the source was not a floppy disk or hard disk, but the console, symbolised by the term CON:, or rather the combination of keyboard and screen.

In that example, the command was:

```
A>copy con: b:notes <RET>
```

The contents of the file immediately appear on the screen again, as in a TYPE-procedure:

```
It is advisable to enter the date and time when
loading the system, so that we later can check
when a file like this was created.
        1 File(s) copied

A>_
```

Note that the text ends with '1 File(s) copied', and that the indication ^Z is no longer displayed. This indication has been converted into a symbol we cannot see, the so-called end-of-file symbol.

In order to achieve the reverse, that is, the transfer from a file on a floppy disk or hard disk to the screen, we only have to turn around the order of the two terms in the formula of the command:

```
A>copy b:notes con: <RET>
```

The result is the same as that of a TYPE command. The difference, however, is that a COPY command enables us to use wildcards in the formula (see page 31).

For instance, if we want to have a look at the contents of the files NOTES.1, NOTES.2 and NOTES.3 on the disk in drive B, the following formula would suffice:

```
A>copy b:notes.* con: <RET>
```

In order to print a file on paper, we can use a COPY command as well. Instead of CON: we type the indication PRN: (for Printer). Thus:

```
A>copy b:notes prn: <RET>
```

Assuming that the printer is connected to the computer and that it is turned on, the text of the file will be printed on paper.

We can also skip a stage. Suppose we want to write a piece of text, a small letter or anything else without storing it in a file. In that case we can combine the COPY CON: command with the term PRN:. Then the formula is:

```
A>copy con: prn: <RET>
```

If we start proceedings as we did on page 50, the same will then appear on the screen. However, nothing is stored now. The text we pass on to the computer by striking the RETURN key is immediately printed on paper. The machine is in fact working like a telex.

While working on the computer, we have become acquainted with several terms from the vocabulary of the central processing unit. We now know that the term for the channel to which a printer can be connected is PRN:. This is an agreement, once made. And there are more agreements like this.

A list of agreements with regard to the input and output channels that are in operation under PC-DOS and MS-DOS looks like this:

CON: - for Console. The combination of screen and keyboard, also known as the terminal. This is both an input channel (keyboard) and an output channel (screen).

PRN: - for Printer. The type of printer is not relevant.

LPT1:, LPT2: and LPT3: - for Line printer. The three channels for parallel output. In computerese, these are referred to as Centronics parallel ports. These channels are only suited for printers.

COM1: and COM2: - for Communications. These are two channels for serial input and output, also known as RS232 ports. They may also be used for a modem, a device which enables the computer to communicate with other computers.

NUL: - the so-called Nul device. A non-existent channel, designed for avoiding the results of a command. At the indication of this channel as source or destination, the system simulates the procedure without actual transport of data taking place.

We can compose some standard formulas for input or output transactions, in the same way as the COPY commands we have met so far. These are:

1:	COPY CON: [x:]<file>	= output from keyboard to disk
2:	COPY CON: PRN:	= output from keyboard to printer
3:	COPY CON: COMa:	= output from keyboard to serial channel
4:	COPY CON: NUL:	= output from keyboard to the nul device
5:	COPY [x:]<file> CON:	= output from disk to screen
6:	COPY [x:]<file> PRN:	= output from disk to printer
7:	COPY [x:]<file> COMa:	= output from disk to serial channel
8:	COPY [x:]<file> NUL:	= output from disk to the nul device
9:	COPY COMa: [x:]<file>	= input from serial channel to disk
10:	COPY COMa: PRN:	= input from serial channel to printer
11:	COPY COMa: CON:	= input from serial channel to screen
12:	COPY COMa: NUL:	= input from serial channel to nul device

In this summary '[x:]' refers to a disk drive, if any, and '<file>' to the name of a file, 'a' to the numeral 1 or 2, and 'serial channel' to a destination or a source that is connected to a serial channel.

3.8: Comparing (2) - COMP

We have now acquired enough information to test the COMP procedure (for Compare). Its function is to check whether or not two files are identical. To do so, we first type the following:

```
A>comp <RET>
Enter primary file name
```

Suppose we want to compare the files TEST, on the diskette in drive A, and TEXT, on the diskette in drive B. We then proceed as follows:

```
Enter primary file name
test <RET>
Enter 2nd file name or drive id
b:text <RET>
```

We could have typed the formula of the command as follows:

```
A>comp test b:text <RET>
```

In any case, we get the opportunity to insert the disk(s) with the files in question in the indicated drive(s). As soon as this has been done, striking any key will start the procedure.

```
Insert diskette(s) with files to compare
and strike any key when ready
```

If the files are not the same size (in bytes), we receive this message:

```
Files are different sizes
```

If the sizes are the same, the files are compared with each other, byte by byte. If the contents are identical, we receive the message:

```
Files compare ok
```

If there turns out to be a difference, we see something like:

```
Compare error at offset B73
File 1 = 3F
File 2 = 07
```

The procedure is automatically halted as soon as ten differences are found:

```
10 Mismatches - aborting compare
```

But whatever the results, we are always asked:

```
Compare more files (Y/N)?_
```

3.9: Deleting - DEL or ERASE

ERASE and DEL (for Delete) are drastic and therefore dangerous commands. Their task is to remove one or more files from a floppy disk or hard disk. The two commands function exactly the same.

An example: suppose we want to remove the file TEXT from the disk in drive A. In that case we type:

```
A>erase text <RET>
```

Or, saving us two key strokes:

```
A>del text <RET>
```

The action is not confirmed - only a new prompt appears. By means of a DIR command, we can ascertain that the file in question has been removed from the directory of the disk in drive A.

By using wildcards (see page 31) in the DEL or ERASE command, we can remove two or more files at the same time.

For instance, if we want to remove the files without an extension from the disk in drive B, we type:

```
A>del b:*. <RET>
```

If we want to delete all files, regardless of the name or the extension, we give the following command:

```
A>del b:*.* <RET>
```

After this command the system is concerned to know if we are sure, asking:

```
Are you sure (Y/N)?_
```

Now the command will only be executed if we type a 'Y' or 'y'. Checking the results by means of DIR command shows that nothing is left on the disk:

```
A>dir b: <RET>
File not found
```

DEL and ERASE commands are dangerous because they are irreversible. The operating system has no instruments to revoke such a command.

Nevertheless, a deleted file is not immediately removed from the disk. The bytes (see page 32) remain in the tracks and sectors (see page 26) until they are overwritten by the contents of another file.

There are utilities available which can reconstruct a file deleted by accident. An example is QU (for Quick Unerase), part of the so-called Norton Utilities.

It is sensible to add such a utility to the operating system and have it at your disposal every time you might need it.

3.10: Checking (2) - CHKDSK

As we have already seen on page 42, the CHKDSK command not only shows the available storage space of a disk, but also the label, if any, and the date and time it was created.

CHKDSK has even more to offer. Let's submit our disk with NOTES and the other files we created to a CHKDSK command. The result could be as follows:

```
A>chkdsk b: <RET>

   362496 bytes total disk space
    23552 bytes in 3 hidden files
    23729 bytes in 7 user files
     5120 bytes in bad sectors
   310095 bytes available on disk

   655360 bytes total memory
   630784 bytes free

A>_
```

We can extend a CHKDSK command with a request to check whether a certain file is cut into two or more parts on the disk. To achieve this, we type:

```
A>chkdsk b:notes <RET>
```

Initially we see a listing such as the one above. Next, we are provided with the information requested:

```
B:\NOTES
   contains 2 non-contiguous blocks.

A>_
```

We may use wildcard symbols in the formula of the command (see page 31).

If we want to check all files on the disk in drive B, we type:

```
A>chkdsk b:*.* <RET>
```

'Non-contiguous' means that a file is stored on the disk (invisible to us) in two or more parts.

If there are many non-contiguous files on a disk, we run the risk that the computer may literally lose track. Therefore, it is advisable to transfer the files to another disk regularly. In order to so, we use the COPY *.* command, because it copies the files one by one and strings them together again.

If we check the disk of destination in the same way, we see:

```
All specified file(s) are contiguous.
```

In the 1-versions of the system (these days hardly ever used), the CHKDSK program automatically corrected all the storage errors and other inconsistencies found in the FAT (see page 47). Usually we did not notice this process, or we saw something like:

```
23748 bytes disk space freed
```

CHKDSK in the 2- and 3-versions of the system provides us with two more options.

First, we can order CHKDSK to report any error or uncertainty so that we can decide what to do with it ourselves. In that case we should add the parameter /f (for fix) to the command:

```
A>chkdsk b:*.*/f <RET>
```

It is possible that we receive a message like the following:

```
B:\NOTES
   Contains invalid cluster, file truncated
```

This means that a part or link of the file is damaged beyond repair. CHKDSK has cut off the file at that point and corrected the data in the FAT.

Normally, another message follows, for instance:

```
008 lost clusters found in 002 chains
Convert lost clusters to files (Y/N)? _
```

In other words, a number of bytes have been found on the disk which are not mentioned in the FAT and which are probably parts of the damaged (truncated) and cut off file.

If we type a 'y', CHKDSK will convert each block ('chain) into a provisional file with the name FILExxxx.CHK, in which xxxx is a number like 0001. Such a provisional file is included in the directory. By means of a TYPE command we can check whether it can be used again.

On the other hand, if we type an 'n', CHKDSK will take no action.

In the second place, as we saw on page 33, we may add the parameter /v to a CHKDSK command.

As a result of this, we are first shown all the files on the disk in question, including the hidden files, if any, followed by more detailed information on the various errors and inconsistencies found on the disk.

Unlike the other formulas of the CHKDSK command, we can redirect the information received in this form to a separate file or printer channel (see page 140). For instance:

```
A>chkdsk b:*.*/v >lpt1: <RET>
```

Assuming a printer is connected to the computer and working, it will print the information.

If we suspect that a disk is damaged, it is wise to give a CHKDSK command followed by the parameter /v first, to get a round-up of the problems that we may face. After that, we repeat the CHKDSK command, but this time with the parameter /f. At that time, we will know where and how to act.

3.11: Reconstructing - RECOVER

By means of the utility RECOVER, we can reconstruct a damaged file more rapidly than by CHKDSK (see previous pages).

Assume that our NOTES file on the disk in drive B has been damaged. We have to give the following command:

```
A>recover b:notes <RET>
```

The first reaction from the computer is:

```
Press any key to begin recovery of the
file(s) on drive B:
```

As soon as we strike any key to start the procedure, RECOVER is going to check the file sector by sector. The sectors that prove to be undamaged are stored in a temporary file. Damaged sectors are left aside but are marked as bad sectors in the FAT (see page 47).

Once all the sectors have been checked, RECOVER converts the temporary file into a final one, giving it the name we have chosen. Next, we receive as message like this:

```
1157 of 1157 bytes recovered

A>_
```

We may include wildcards in the command, but only the first file to which the formula applies will be recovered. This may save us, however, some time if the file name is long and difficult.

RECOVER may also come in handy when CHKDSK is useless, for instance when the FAT is irreparably damaged. In that case (and only then) we merely indicate the drive. Thus:

```
A>recover b: <RET>
```

At first, the procedure is the same. Then a message like this follows:

```
7 file(s) recovered

A>_
```

If we display the directory of the disk in question by means of a DIR command, the files turn out to have been given new names, namely FILExxxx.REC, in which xxxx is a number like 0001, plus the date and the time of this procedure.

What remains of the original directory is the size of the files in bytes. Now we must try to find out which file is which, using TYPE commands, and to give each file its old name back, using REN commands. If there are a great number of files involved, this job is no fun.

3.12: Protecting files - ATTRIB

In the 2-versions of the system, for instance, PC-DOS version 2.10, or MS-DOS version 2.11, we can only protect files on a disk against accidental damage or erasure by means of closing the small, square notch on the right side of the disk. And there is no way to protect files on a hard disk.

With the introduction of the 3-versions of MS-DOS and PC-DOS, however, we have a facility at our disposal to protect any file on both a floppy disk and a hard disk against damage or destruction - with the exception of a FORMAT procedure. This facility consists of a utility called ATTRIB (for Attribute).

Suppose we want to provide the file NOTES on the disk in drive B with such a protection. Assuming that the ATTRIB.EXE file is on the system disk in drive A, we type the following command:

```
A>attrib +r b:notes <RET>

A>_
```

We now can TYPE the file, or COPY it. If we try to erase the file, however, it turns out to have become impervious:

```
A>del b:notes <RET>
Access denied
```

Note that the command includes the letter 'r' preceded by a plus sign.

In a similar way, we can cancel the protection of the file by typing '-' instead of '+'. Thus:

```
A>attrib -r b:notes <RET>
```

If we want to be informed on the status of a file, in other words, whether is it protected or not, we type the following command:

```
A>attrib b:notes <RET>
```

This may result in an 'R' (for Read only) on the screen:

```
R        B:\NOTES
```

If the file is not protected, the 'R' is not shown. Thus:

```
         B:\NOTES
```

By means of wildcards (see page 31) we can check, add or remove the protection of several files at the same time. For instance:

```
A>attrib b:*.* <RET>
```

4: CHARACTERS AND SYMBOLS

As we mentioned on page 32, all instructions, texts and data we pass on to the computer have to be converted into zeros and ones in accordance with a specific coding system, before the CPU (see page 13) can do anything with them. Such a system consisting of no more than two values is called a binary system. In the binary system, the numeral 2 is represented by 10, 3 by 11, 4 by 100, 5 by 101, 6 by 110, 7 by 111, 8 by 1000, 9 by 1001, 10 by 1010, etc.

The use of the binary system in telecommunication is not new. In the 19th century the Morse code came into fashion when telegraphy was introduced. This was a method to convert the letters A to Z and the numerals 0 to 9 into a combination of short and long electric pulses. The well-known distress signal SOS, for instance, consists of three short pulses (meaning the letter S), three long pulses (the letter O), and again three short pulses.

For a very long time, the Morse code was an adequate method to send messages by means of electronics, but it also involved some disadvantages. Punctuation marks were left aside, no distinction was made between capitals and lower case, and some characters needed no less than five pulses. When teletype machines like the telex and later printing machines of the bigger computer systems were introduced, the Morse code was no longer adequate.

Initially a chaos of various coding systems arose, with the result that machines of different makes were hardly or not at all compatible. Between 1968 and 1977 this chaos came to an end when the ANSI (for American National Standards Institute) designed and developed a standard system named ASCII (American Standard Code for Information Interchange). Since then, it has been adopted by most of the manufacturers of telexes and computers.

The ASCII system consists of a list of 128 codes, from 0000000 (that is, 0 in the decimal system) to 1111111 (that is, 128). This list contains, of course, all the letters in the alphabet, both in capitals and in lower case, as well as the numerals 0 to 9, plus a large number of punctuation marks, but also various codes for certain instructions, such as the instruction to the printer to pull in a new sheet of paper (FF, for form feed), or the instruction to position the printer head on the next line (LF, for line feed).

In the middle of the seventies, when we were starting to work with early types of microcomputers, every letter or instruction had to be passed on to the computer very accurately in groups of seven zeros and ones. Fortunately, we do not have to do this anymore. The operating system is equipped with some sort of intermediate code, based on the hexadecimal system. The first ten numerals (0-9) are the same as in the decimal system. The value 10 is represented by the letter A, 11 by B, 12 by C, 13 by D, 14 by E and 15 by F.

If we continue counting, the figures will consist of two numbers beginning with a '1', like in the decimal system. For instance, 16 in the decimal system is 10 in the hexadecimal system, 17 is 11, 18 is 12 and so on, up to 25 which is 19. Next, we see the letters again: 26 is 1A, 27 is 1B, 28 is 1C, 29 is 1D, 30 is 1E, and 31 is 1F. The next hexadecimal number is 20 which is 32 in the decimal system. If we look up the decimal number 128, we will find the value 7F.

In this way, all 128 codes from the ASCII system are defined by a hexadecimal figure consisting of two characters. In other words, whatever code from this list of 128 we wish to pass on, we only need two key strokes.

4.1: The ASCII table

dec.	hex.	binary	symb.	name	dec.	hex.	binary	symb.	name
00	00	0000000	^@	NUL	30	1E	0011110	^^	RS
01	01	0000001	^A	SOH	31	1F	0011111	^_	US
02	02	0000010	^B	STX	32	20	0100000		SP
03	03	0000011	^C	ETX	33	21	0100001	!	
04	04	0000100	^D	EOT	34	22	0100010	"	
05	05	0000101	^E	ENQ	35	23	0100011	#	
06	06	0000110	^F	ACK	36	24	0100100	$	
07	07	0000111	^G	BEL	37	25	0100101	%	
08	08	0001000	^H	BS	38	26	0100110	&	
09	09	0001001	^I	HT	39	27	0100111	'	
10	0A	0001010	^J	LF	40	28	0101000	(	
11	0B	0001011	^K	VT	41	29	0101001	)	
12	0C	0001100	^L	FF	42	2A	0101010	*	
13	0D	0001101	^M	CR	43	2B	0101011	+	
14	0E	0001110	^N	SO	44	2C	0101100	,	
15	0F	0001111	^O	SI	45	2D	0101101	-	
16	10	0010000	^P	DLE	46	2E	0101110	.	
17	11	0010001	^Q	DC1	47	2F	0101111	/	
18	12	0010010	^R	DC2	48	30	0110000	0	
19	13	0010011	^S	DC3	49	31	0110001	1	
20	14	0010100	^T	DC4	50	32	0110010	2	
21	15	0010101	^U	NAK	51	33	0110011	3	
22	16	0010110	^V	SYN	52	34	0110100	4	
23	17	0010111	^W	ETB	53	35	0110101	5	
24	18	0011000	^X	CAN	54	36	0110110	6	
25	19	0011001	^Y	EM	55	37	0110111	7	
26	1A	0011010	^Z	SUB	56	38	0111000	8	
27	1B	0011011	^[	ESC	57	39	0111001	9	
28	1C	0011100	^\	FS	58	3A	0111010	:	
29	1D	0011101	^]	GS	59	3B	0111011	;	

dec.	hex.	binary	symb.	name	dec.	hex.	binary	symb.	name
60	3C	0111100	<		94	5E	1011110	^	
61	3D	0111101	=		95	5F	1011111	_	
62	3E	0111110	>		96	60	1100000	`	
63	3F	0111111	?		97	61	1100001	a	
64	40	1000000	@		98	62	1100010	b	
65	41	1000001	A		99	63	1100011	c	
66	42	1000010	B		100	64	1100100	d	
67	43	1000011	C		101	65	1100101	e	
68	44	1000100	D		102	66	1100110	f	
69	45	1000101	E		103	67	1100111	g	
70	46	1000110	F		104	68	1101000	h	
71	47	1000111	G		105	69	1101001	i	
72	48	1001000	H		106	6A	1101010	j	
73	49	1001001	I		107	6B	1101011	k	
74	4A	1001010	J		108	6C	1101100	l	
75	4B	1001011	K		109	6D	1101101	m	
76	4C	1001100	L		110	6E	1101110	n	
77	4D	1001101	M		111	6F	1101111	o	
78	4E	1001110	N		112	70	1110000	p	
79	4F	1001111	O		113	71	1110001	q	
80	50	1010000	P		114	72	1110010	r	
81	51	1010001	Q		115	73	1110011	s	
82	52	1010010	R		116	74	1110100	t	
83	53	1010011	S		117	75	1110101	u	
84	54	1010100	T		118	76	1110110	v	
85	55	1010101	U		119	77	1110111	w	
86	56	1010110	V		120	78	1111000	x	
87	57	1010111	W		121	79	1111001	y	
88	58	1011000	X		122	7A	1111010	z	
89	59	1011001	Y		123	7B	1111011	{	
90	5A	1011010	Z		124	7C	1111100	|	
91	5B	1011011	[		125	7D	1111101	}	
92	5C	1011100	\		126	7E	1111110	~	
93	5D	1011101	]		127	7F	1111111		DEL

4.2: Control instructions

When we take a close look at the ASCII table, the first thing that strikes us is that the first 33 codes, with the decimal values 0 to 32, and the last one with the decimal value 127 are followed by a term of 2 or 3 letters.
These terms are abbreviations, meaning:

NUL = null	DC1 = device control 1
SOH = start of heading	DC2 = device control 2
STX = start of text	DC3 = device control 3
ETX = end of text	DC4 = device control 4
EOT = end of transmission	NAK = negative acknowledgement
ENQ = enquiry	SYN = synchronous idle
ACK = acknowledge	ETB = end of transmission block
BEL = bell	CAN = cancel
BS = backspace	EM = end of medium
HT = horizontal tabulation	SUB = substitute
LF = line feed	ESC = escape
VT = vertical tabulation	FS = file separator
FF = form feed	GS = group separator
CR = carriage return	RS = record separator
SO = shift out	US = unit separator
SI = shift in	SP = space
DLE = data link escape	DEL = delete

In this list of codes we also see that the column 'symbol' does not state any symbols we can find on the keyboard. This is due to the fact that these ASCII codes do not represent a specific character, but an instruction. On page 18 we have already seen an example: the instruction to place the cursor one place back and erase the character, if any, from that position.

As we have learned, we can pass on this instruction to the operating system in two ways. First, by striking the key on the top row that is marked BS or showing an arrow pointing to the left, and second, by means of a so called control instruction.

A control instruction consists of striking the CTRL key, keeping it depressed, and hitting the relevant character, in this case the capital or lower case H. Such a control instruction is indicated by a circumflex (a 'hat'), followed by the character in question. Thus: ^H.

If we now look up the ^H symbol in the ASCII table, we see that the corresponding decimal and hexadecimal values are 08. And indeed, the abbreviation belonging to it is BS.

Another example is ESC, for Escape, which we can find in the ASCII table with the code having the decimal value 27, and the hexadecimal value 1B. On page 18 we have already seen one of its effects. In this example, too, it concerns an instruction we can pass on to the operating system either by means of a control instruction (namely ^[), or by a special key (ESC).

A third example is the RETURN key. Striking this key also results in an instruction to be passed on to the operating system (see page 16). The corresponding term in the ASCII table is CR (for Carriage Return), with the decimal value 13 and the hexadecimal value 0D.

Control instructions are indispensable tools of the operating system. In general, they serve to make clear to the system that a certain operation has to be carried out. In the long run we will find out that some of these control instructions will be used regularly and others hardly ever.

The most important control instructions are:

^C - the cancel instruction. Cancels (if possible) the last issued command or ends the current program. The same effect may be achieved by means of the instruction ^<BREAK> or ^<SCROLL LOCK>.

^G - the bell instruction. Cannot be given from the keyboard directly, but only as a part of a program file.

^H - the backspace instruction. Moves the cursor one place back and removes the character from that position. This instruction may also be given by means of the key on the top row of the keyboard marked BS or showing an arrow pointing to the left.

^I - the tab instruction. Moves the cursor a fixed number of places (usually eight) to the right. This instruction may also be given by means of the key to be found on the left of the keyboard marked with the term TAB, or showing two arrows pointing in opposite directions.

^J - the line feed instruction. Places the cursor on the next line, enabling us to continue typing a command. The same effect can be achieved by means of the instruction ^<RET>.

^L - the form feed instruction. Passes on the instruction that the printer has to insert a new sheet of paper.

^M - the carriage return instruction. A remainder of the telex. Nowadays this instruction results in the last-issued command being passed on. The same effect may be achieved by striking the RETURN key.

^P - the print instruction. Starts or terminates output to the printer of characters appearing on the screen. The same effect may be achieved by the instruction ^<PRT SC>.

^S - the pause instruction. Interrupts the flow of characters on the screen during a TYPE procedure. See page 51. The same effect may be achieved by the instruction ^<NUM LOCK>.

^Z - the end-of-file instruction. Tells the operation system when the end of the file has been reached. See page 50.

^<ALT><DEL> - the reset instruction. Cancels all current procedures, empties the memory and reloads the system. Actually, this instruction does not belong in this list since it is not based on the ASCII system. Instead, it is a BIOS command (see page 25), implemented by IBM. At the time, IBM decided not to equip its Personal Computer with a RESET button, but rather to oblige the user to strike three keys at the same time. All manufacturers of 'IBM compatible' computers have endorsed this decision, although there is also a number of manufacturers that have added an extra reset button to the computer.

4.3: Reproducing (3) - DEBUG

On page 67 we saw that all characters we pass on to the computer are first converted by the system into a hexadecimal number of two characters at the most. We can demonstrate this by means of a system utility named DEBUG.

To do so, we insert a disk containing the DEBUG.COM file in drive A and a disk with a copy of our NOTES file in drive B. Then we type the following:

```
A>debug b:notes <RET>
-
```

The system's answer is a hyphen. This is the DEBUG prompt.

Then we type a D or d, for dump, and strike the RETURN key. As a result, the following table of values and letters will appear on the screen:

```
:0100  49 74 20 69 73 20 61 64-76 69 73 61 62 6C 65 20  It is advisable
:0110  74 6F 20 65 6E 74 65 72-20 74 68 65 20 64 61 74  to enter the dat
:0120  65 20 61 6E 64 20 74 69-6D 65 20 77 68 65 6E 0D  e and time when.
:0130  0A 6C 6F 61 64 69 6E 67-20 74 68 65 20 73 79 73  .loading the sys
:0140  74 65 6D 2C 20 73 6F 20-74 68 61 74 20 77 65 20  tem, so that we
:0150  6C 61 74 65 72 20 63 61-6E 20 63 68 65 63 6B 0D  later can check.
:0160  0A 77 68 65 6E 20 61 20-66 69 6C 65 20 6C 69 6B  .when a file lik
:0170  65 20 74 68 69 73 20 77-61 73 20 63 72 65 61 74  e this was creat
```

As we can see, the table consists of three columns. The first indicates the exact place (the so-called address) of the next line of characters in the computer's memory. These characters can be found in the second column, stated in hexadecimal notations. The third column provides us with the translation in ordinary ASCII characters.

In order to display the next block of characters, we again type a D or d, followed by striking the RETURN key. In this example the result will be:

```
:0180  65 64 2E 0D 0A 1A 00 00-00 00 00 00 00 00 00 00  ed..............
:0190  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
:01A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
:01B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
:01C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
:01D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
:01E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
:01F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Note that the last character before the zeros take over is 1A. This is the end of file sign we became acquainted with on the previous page.

We can leave the DEBUG program by typing a Q or q (for quit), followed by the usual stroke of the RETURN key, after which the system prompt reappears.

4.4: Tracing - FIND

On pages 51, 54 and the previous page we became acquainted with various methods to display the contents of a file on the screen. These methods are not adequate if we want to trace a certain character or a string (a series of characters or a fragment), especially when the file is rather large. In that case we may better use the FIND utility.

For instance: suppose we want to find the lines in our NOTES file containing the word 'date' (see page 52). Assuming that the FIND.EXE file is on the disk in drive A, we type the following command:

```
A>find 'check' b:notes <RET>
```

Then the following reaction appears on the screen:

```
---------- b:notes
loading the system so that we later can check
A>_
```

If we also want to know in which line the character or string can be found, we add the parameter /n to the command. Thus:

```
A>find/n 'check' b:notes <RET>
```

```
---------- b:notes
[2]loading the system so that we later can check
A>_
```

If we do not want to view the line itself but only want to know how many times the character or string occurs in the file, we add the parameter /c to the command. Thus:

```
A>find/c 'check' b:notes <RET>
```

```
---------- b:notes: 1
A>_
```

On the other hand, if we want to have the line(s) displayed or counted in which the character or string does not occur, we add the parameter /v to the command. Thus:

```
A>find/c/v 'check' b:notes <RET>
```

```
---------- b:notes: 3
A>_
```

By means of a single FIND instruction, whether or not extended with parameters, we can even inspect two or more files at the same time. For instance:

```
A>find/c 'check' b:notes b:alphabet <RET>
```

4.5: The keyboard (2)

Both the ASCII system and the keyboard lay-out are of American origin. Consequently, the choice and lay-out are based on American standards. But not all characters are equally important outside the United States. For instance, in the United Kingdom they prefer the £ character on the keyboard, and in France the cedilla (ç), the diaresis (¨), and the section mark (§).

These characters can be found in the extended character set, as we will see on the next page, but the use of the ALT key and the numerals on the numeric keypad is rather time-consuming. That is why IBM and other manufacturers with branches in Europe have equipped their computer systems with special versions of the keyboard. For instance, if we work with a PC of British origin, the keyboard contains the pound sterling symbol on the 3 key, instead of the American # sign.

Moreover, the additional version of MS-DOS or PC-DOS contains a number of .COM files, such as KEYBFR, KEYBGR, KEYBIT, KEYBSP, and KEYBUK. These files contain a small program that, once started, takes over the interaction between the keyboard and the part of the ROM BIOS (see page 25) stating the effect of every key stroke.

If we want to get the £ sign on the screen after hitting this key, we first have to install the keyboard. In that case we have to use the KEYBUK utility, for Keyboard UK. Assuming that the system disk is in drive A and that it contains the KEYBUK.COM file, we activate the utility as follows:

```
A>keybuk <RET>
```

There is no confirmation whatsoever, only the prompt appears again. And indeed, if we now hit the SHIFT and 3 keys, the £ sign appears on the screen.

Moreover, the system has not forgotten the lay-out according to the original, American standard. So if we want to see the original (American) result of a key stroke at a certain point, we need not load the system again.

For we have a so-called toggle, or switch, at our disposal. It is activated by hitting the CTRL key and the ALT key at the same time, followed by striking the function key F1. The American keyboard lay-out is put in operation right away. To demonstrate, we press the SHIFT and 3 keys. And again the American # sign appears instead of the pound sterling symbol.

If we then want to switch to the British keyboard lay-out, we have to hit the CTRL key and the ALT key again, followed by the F2 key.

We can install a French keyboard lay-out in a similar way as well.

Assuming that the disk containing the KEYBFR.COM is in drive A, we have to type the following command:

```
A>keybfr <RET>
```

And of course, we can install the German, Italian and Spanish keyboard layout by means of the KEYBGR, KEYBIT, and KEYBSP utilities, respectively. But although we can switch between a European and the American keyboard lay-out, it is not possible to switch from one European keyboard lay-out to another.

4.6: The extended character set

At the end of the seventies, when IBM started developing the PC, the collection of characters in the ASCII system soon turned out to be inadequate. Therefore, such international companies as IBM considered it essential to adopt certain European characters, as we saw on the previous page.

In addition, there was a need for a number of technical and scientific symbols and various special signs for producing lines, blocks and rasters on the screen.

This problem was solved by an unofficial extension of the ASCII table with another 128 characters. This was possible because the binary code of the first 128 characters only contained 7 units (bits). The eighth bit completing the byte was merely used for control purposes, but when a computer is used in the normal way, this extra bit is not needed.

IBM made use of this space to double the number of available characters, so that its total amounted to 256. The extra 128 characters became known as 'the extended character set'. The next two pages show a survey of this set.

Another problem for IBM was the question how these extra characters should be called up. The answer was the addition of an extra key to the keyboard, marked with the term ALT, for Alternate. In combination with the keys on the numeric keypad that can be found on the right of the keyboard, the ALT key can be used to produce all extra characters of the 'extended character set' on the screen.

For instance, the decimal value of the £ sign is 156, as the list shows. Now we hit the ALT key, keep it depressed and type a 1, a 5 and a 6 on the numeric keypad. As soon as we release the ALT key, the character in question appears on the screen.

We can also enter such a sign in a text file in a similar way. To demonstrate, we create a file named POUND by means of a COPY CON: procedure (see page 50). Thus:

```
A>copy con: b:pound <RET>
the character for the pound sterling is
```

Once we have reached this stage, we type the £ sign in the way mentioned above. Then we hit the F6 key or type ^Z. Finally we press the RETURN key.

If the instruction has been carried out properly, the disk in drive B contains the POUND file including the character in question. This can be ascertained by means of a TYPE procedure:

```
A>type b:pound <RET>
the character for the pound sterling is £
A>_
```

We should not refer to this set of extra characters as ASCII signs, since the ANSI bureau had nothing to do with it.

On the other hand, the influence of IBM's Personal Computer and all its compatibles has become so indispensable that, in practice, this extension of the ASCII set has become a standard. All other PC manufacturers were wise enough to adopt this in due course.

The extended character set

dec.	hex.	binary	symbol	dec.	hex.	binary	symbol
128	80	10000000	Ç	158	9E	10011110	₧
129	81	10000001	ü	159	9F	10011111	ƒ
130	82	10000010	é	160	A0	10100000	á
131	83	10000011	â	161	A1	10100001	í
132	84	10000100	ä	162	A2	10100010	ó
133	85	10000101	à	163	A3	10100011	ú
134	86	10000110	å	164	A4	10100100	ñ
135	87	10000111	ç	165	A5	10100101	Ñ
136	88	10001000	ê	166	A6	10100110	ª
137	89	10001001	ë	167	A7	10100111	º
138	8A	10001010	è	168	A8	10101000	¿
139	8B	10001011	ï	169	A9	10101001	⌐
140	8C	10001100	î	170	AA	10101010	¬
141	8D	10001101	~	171	AB	10101011	½
142	8E	10001110	Ä	172	AC	10101100	¼
143	8F	10001111	Å	173	AD	10101101	¡
144	90	10010000	É	174	AE	10101110	«
145	91	10010001	æ	175	AF	10101111	»
146	92	10010010	Æ	176	B0	10110000	░
147	93	10010011	ô	177	B1	10110001	▒
148	94	10010100	ö	178	B2	10110010	▓
149	95	10010101	|	179	B3	10110011	│
150	96	10010110	û	180	B4	10110100	┤
151	97	10010111	ù	181	B5	10110101	╡
152	98	10011000	ÿ	182	B6	10110110	╢
153	99	10011001	Ö	183	B7	10110111	╖
154	9A	10011010	Ü	184	B8	10111000	╕
155	9B	10011011	¢	185	B9	10111001	╣
156	9C	10011100	£	186	BA	10111010	║
157	9D	10011101	¥	187	BB	10111011	╗

dec.	hex.	binary	symbol	dec.	hex.	binary	symbol
188	BC	10111100	╝	222	DE	11011110	▐
189	BD	10111101	╜	223	DF	11011111	▀
190	BE	10111110	╛	224	E0	11100000	α
191	BF	10111111	┐	225	E1	11100001	β
192	C0	11000000	└	226	E2	11100010	Γ
193	C1	11000001	┴	227	E3	11100011	π
194	C2	11000010	┬	228	E4	11100100	Σ
195	C3	11000011	├	229	E5	11100101	σ
196	C4	11000100	─	230	E6	11100110	μ
197	C5	11000101	┼	231	E7	11100111	τ
198	C6	11000110	╞	232	E8	11101000	Φ
199	C7	11000111	╟	233	E9	11101001	Θ
200	C8	11001000	╚	234	EA	11101010	Ω
201	C9	11001001	╔	235	EB	11101011	δ
202	CA	11001010	╩	236	EC	11101100	∞
203	CB	11001011	╦	237	ED	11101101	ϕ
204	CC	11001100	╠	238	EE	11101110	$\in$
205	CD	11001101	═	239	EF	11101111	$\cap$
206	CE	11001110	╬	240	F0	11110000	$\equiv$
207	CF	11001111	╧	241	F1	11110001	$\pm$
208	D0	11010000	╨	242	F2	11110010	$\geq$
209	D1	11010001	╤	243	F3	11110011	$\leq$
210	D2	11010010	╥	244	F4	11110100	⌠
211	D3	11010011	╙	245	F5	11110101	⌡
212	D4	11010100	╘	246	F6	11110110	$\div$
213	D5	11010101	╒	247	F7	11110111	$\approx$
214	D6	11010110	╓	248	F8	11111000	°
215	D7	11010111	╫	249	F9	11111001	∙
216	D8	11011000	╪	250	FA	11111010	·
217	D9	11011001	┘	251	FB	11111011	$\surd$
218	DA	11011010	┌	252	FC	11111100	^{n}
219	DB	11011011	█	253	FD	11111101	^{2}
220	DC	11011100	▄	254	FE	11111110	■
221	DD	11011101	▌	255	FF	11111111	

5: USING A HARD DISK

A hard disk (also known as a fixed disk or Winchester disk) is a storage medium that takes up a permanent place in the computer. It may be a standard part of the computer, as in IBM's PC XT, but it could also be added later. In that case there are three possibilities: external, in a special box, or internal, instead of a floppy disk drive, or in the form of a special card to be inserted in the slots in the system cabinet specially designed.

Whatever the type or make, the hard disk is usually enclosed in a sealed housing which contains various separate disks plus the relevant recording heads. It may take the place of a floppy disk drive, or it comes in the form of a so-called hard disk card.

The main advantage of a hard disk over floppy disks is the enormous storage capacity. The hard disk of IBM's PC XT and similar machines contain at least ten and usually twenty or thirty megabytes. In other words, some thirty to eighty times the data of a floppy disk. The hard disk of the PC AT and similar machines as well as the new IBM PS/2 version even have a considerably larger capacity, varying from 30, 40 and 70 to even 110 megabytes.

Apart from this huge storage capacity, the hard disk offers us the convenience of the programs and files we regularly use being permanently available: we simply store them on the hard disk.

In addition, a computer equipped with a hard disk responds much faster than a machine with floppy disks, especially when loading and storing files. Only the so-called RAM disk (see page 128) operates even faster, but with a RAM-disk we are, on the other hand, defenceless against a breakdown.

A final advantage of a hard disk is that it can save space, provided that the computer 'boots from drive C'. In other words, if the system can be loaded from the hard disk (usually indicated by drive C). We will return to this on page 82. If this is the case and if we work with a kind of PC AT computer, that is, a big, rather unmanageable machine, it may be practical, with regard to the space, to place only the keyboard and the screen on the desk and the system cabinet containing the disk drives and those sort of things under it, upright, with the power switch within reach.

A disadvantage of the hard disk is that its housing is more vulnerable than that of a floppy disk. As a result of, for instance, someone bumping against the system cabinet or a power blackout, a so-called head crash may occur. This is a collision between a registration head and part of the writing surface, which usually causes the stored data to become damaged or lost.

The more the technological ingenuity is developing, the better the disks will be protected against head crashes, but they will never be avoided entirely. Therefore, it is essential to copy the contents of the hard disk regularly. We should bear in mind, however, that there is no simple and inexpensive method to do this. The procedures provided by the system are slow and difficult and require a large number of floppy disks.

A sufficient, but yet rather expensive solution is to add a so-called tape streamer to the machinery. It normally enables us to copy or reconstruct the contents of the entire hard disk in less than ten minutes but it is almost as expensive as the hard disk itself. Fortunately, prices of these two facilities are gradually decreasing.

5.1: Installing

Before we can actually use a hard disk, we have to take a few steps. First, we have to install the hard disk so that the system can take control, and next we have to format it so that the system can store data on it.

The procedure to install the hard disk depends on its make and type. Usually, we need to run a certain program. To install the hard disk in the PC XT and PC AT, for instance, we have to run the FDISK.COM program that is supplied with PC-DOS in the versions 2.0 and higher.

If the computer or hard disk is of a different make or type, the dealer or the instruction manual can no doubt make clear what we have to do.

Let's assume we have a PC XT computer. First, we have to load the system (see page 14). Assuming that the system disk containing the FDISK.COM file is in drive A, we type the following command:

```
A>fdisk <RET>
```

As a result, the following list appears on the screen:

```
IBM Personal Computer
Fixed Disk Setup Program Version 3.10
(c) Copyright IBM Corp. 1983, 1985

FDISK Options

Choose one of the following:

     1. Create DOS Partition
     2. Change Active Partition
     3. Delete DOS Partition
     4. Display Partition Data

Enter choice: [_]
```

Such a list of available options is called a menu. When operating the computer, we will come across such menus regularly.

Fortunately, we do not have to make a choice now. Striking the RETURN key is enough - FDISK knows what to do.

Should we want to cancel the procedure before FDISK is put into operation, then we must press the ESC key immediately. Moreover, it is advisable not to experiment with the rest of the options stated on the FDISK menu if we are not completely familiar with the hard disk.

After the RETURN key has been struck, two different texts can appear on the screen. The following is the first of the two possibilities:

```
DOS Partition already created
```

With this message FDISK indicates that the procedure has already been carried out - presumably by the supplier. Then we only have to format the disk.

The second possibility is:

```
Create DOS Partition

Do you wish to use the entire fixed disk
for DOS (Y/N).....................? [_]
```

Again we do not have to worry about the answer. Striking the RETURN key will do. As a result we see:

```
System will now restart
Insert DOS diskette in drive A:
Press any key when ready . . ._
```

If the system disk is still in drive A, we can just press any key. The result is that the system is reloaded.

The next step is to format the disk.

In order to start the procedure we type the following command, assuming that the system disk containing the FORMAT.COM file is in drive A and that the letter C is the indication of the hard disk:

```
A>format c: <RET>
```

We may also add the parameter /s to this command. Thus:

```
A>format c:/s <RET>
```

Adding this parameter results in a copy of the system files being transferred to the hard disk (see page 44) so that it is no longer necessary to have a system disk in drive A to load the system.

```
Press any key to begin formatting drive C:
```

Should we ever see a message like this when this is not the intention, we have to stop the procedure immediately by means of a ^C or ^<BREAK> instruction (see page 71).

Pressing any other key will start the FORMAT procedure irrevocably. For about half an hour we see:

```
Formatting cylinder :
```

Finally, this is followed by a few messages that indicate the end of the procedure, for instance:

```
Format complete
System transferred
```

Next, a survey like the following is displayed:

```
21317632 bytes total disk space
   59930 bytes used by system
   55296 bytes in bad sectors
21102406 bytes available on disk
```

5.2: Loading the system (2)

On page 25 we became acquainted with the term ROM BIOS as a collective name for all codes and instructions stored in the computer permanently. It also includes the instruction, after having turned on the machine or after a RESET procedure (see page 18), to check whether drive A contains a legible disk and, if that is the case to trace the presence of the two 'hidden files' (see page 48) and the COMMAND.COM file.

If this is all found in order, the system is loaded. If not, IBM computers automatically switch to BASIC in the ROM BIOS (see page 25). With other makes of computers, a message will appear on the screen, usually suggesting the insertion of a system disk in drive A after all.

When a computer is being equipped with a hard disk of any type, the ROM BIOS has to be provided with additional instructions at the same time. This is done by means of an extra ROM chip that can be found on the so-called controller card of the hard disk. These additional instructions include data about the hard disk itself, such as the storage capacity.

In addition, they add certain instructions that constitute an intermediate phase in the loading procedure. These instructions in fact mean the addition of an extra procedure: if drive A does not contain a usable disk, the hard disk must be inspected for system files. If they are found, and this is only possible if the hard disk has been formatted with the /s parameter (see page 44), then the system will be loaded from the hard disk. In other words, the computer 'boots from drive C'.

After the date and time messages, a familiar message like the following may then appear on the screen:

```
The IBM Personal Computer DOS
Version 3.30 (C)Copyright International Business
             (C)Copyright Microsoft Corp 1981, 1

C>_
```

The prompt indicates that the hard disk (drive C) has now become active.

The additions to the ROM BIOS, however, do not alter the fact that drive A is always being inspected to contain a disk with usable system files first. This may come in handy, for instance after a crash of the hard disk. Owing to the fact that it is possible to load the system from drive A, we can check whether we can save some of the contents of the hard disk.

Or suppose we want to test another version of MS-DOS or PC-DOS. Then we only have to insert the disk containing the system in drive A and turn on the computer or start a RESET procedure.

We have to bear in mind that we cannot simply replace the system files on the hard disk by those of a more recent version of the system. The problem is that, in every new version, the system files take up more space than in the previous one, as the survey on page 000 proves, and the system requires that the 'hidden files' are stored in the first tracks of the hard disk.

So in that case there is no alternative than to make a copy of the contents of the hard disk (see page 92) and then to re-format it.

5.3: Creating a sub-directory - MD

One of the advantages of the hard disk is that we can store a large number of files, 512 to be exact, but it is obvious that the total size of the files may not exceed the available storage capacity.

This vast number of files forms at the same time its main disadvantage, for how are we to keep track of the situation? In order to display sections of the directory on the screen we can, of course, give a DIR command with the parameter /p. Thus:

```
C>dir/p <RET>
```

If the hard disk contains some hundreds of files, however, we would have to wade through page after page full of names and other data, before we would find that one file we were looking for.

There is a method to maintain a clear survey of the contents on the hard disk. In addition to the normal directory, also known as the main or root directory, we can create sub-directories that can contain an unlimited number of files.

To create a sub-directory, we first give a MKDIR or MD command (for Make Directory), followed by a name to a maximum of ten characters.

Let's create a sub-directory by way of experiment and name it DOS. To do so we type the following:

```
C>md \dos <RET>
```

Note that we have used a backslash (\) in the formula. This character symbolises the root directory. It precedes the name of the sub-directory, in this example DOS, in order to mark the order, or rather the hierarchy.

At first we see nothing unusual - just the prompt. But if we now call up the directory of the hard disk, we see:

```
C>dir <RET>

 Volume in drive C has no label
 Directory of  C:\

COMMAND  COM    23612   1-05-87  12:00a
DOS          <DIR>     12-21-88   1:11p
        2 File(s)  21102406 bytes free
```

Note that DOS has been given a place in the directory, along with date and time entries, as if it were a normal file. However, the indication <DIR> takes up the place of the extension and the size of the file.

When we try to display the file on the screen, it turns out that it is not a normal file:

```
C>type dos <RET>
File not found
```

A sub-directory is handled in about the same way as a disk drive.
For instance, to find out which files are present in a sub-directory, we have to type the entire designation of the sub-directory:

```
C>dir\dos <RET>
```

If the active drive is not C (i.e. the hard disk) but A, for instance, we have to adjust the command as follows:

```
A>dir c:\dos <RET>
```

In both cases the reaction will look something like:

```
 Volume in drive C has no label
 Directory of  C:\dos

.            <DIR>      12-21-88   1:11p
..           <DIR>      12-21-88   1:11p
        2 File(s)  21102406 bytes free
```

Note that the directory just consists of entries both beginning with dots, followed by <DIR>. Apart from those, the sub-directory is empty.

Transferring files is not different from what we are used to with a normal disk drive.
For instance, in order to transfer a copy of the NOTES file on the disk in drive A to the sub-directory of drive C, we type:

```
A>copy notes c:\dos/v <RET>
```

If we then take a look at the directory of drive C, it seems that the copied file has disappeared:

```
A>dir c: <RET>

 Volume in drive C has no label
 Directory of  C:\

COMMAND  COM     23612   1-05-87  12:00a
DOS          <DIR>      12-21-88   1:11p
        2 File(s)  21102406 bytes free
```

On the other hand, the amount of 'bytes free' turns out to have decreased. To find the copy again, we have to address the sub-directory directly:

```
A>dir c:\dos <RET>

 Volume in drive C has no label
 Directory of  C:\dos

.            <DIR>      12-21-88   1:11p
..           <DIR>      12-21-88   1:11p
NOTES              134  12-17-88  10:17a
        3 File(s)  21102406 bytes free
```

Now suppose we want to remove the NOTES file from the sub-directory. If A is the active drive, we should formulate the command as follows:

A>del c:\dos\notes <RET>

If C is the active drive, we type:

C>del \dos\notes <RET>

Note that we typed the \ symbol twice. The first one was to mark the beginning of the chain, the root of the tree so to speak, and the second to separate the name of the sub-directory, in this case DOS, from the name of the file in question, in this case NOTES.

We may use wildcard symbols (see page 31). For instance, if we want to remove all files en bloc from this sub-directory, we can issue the following command:

A>del c:\dos*.* <RET>

Or, if C is the active drive:

C>del \dos*.* <RET>

As usual we are, in both cases, requested confirmation of the instruction (see also page 61):

Are you sure (Y/N)?_

5.4: Removing a sub-directory - RD

We can remove a sub-directory only if it is empty, in other words, if it does not contain files. If that is the case, we issue the command RMDIR or RD (for Remove Directory):

A>rd c:\dos <RET>

Or, if C is the active drive:

C>rd \dos <RET>

In both cases a DIR command shows that the sub-directory has indeed disappeared.

If we try to remove a sub-directory still containing one or more files, the following error message appears on the screen:

Invalid path, not directory,
or directory not empty

5.5: Navigating - CD

In some cases it is impractical, and in other cases even impossible, to operate from the root directory with files in a sub-directory. In those cases it is essential to switch to the level of the relevant sub-directory. To do so, we have to use the CHDIR or CD command (for Change Directory).

Suppose that C is the active drive containing a sub-directory named DOS. In order to switch to the level of DOS we type:

```
C>cd \dos <RET>

C>_
```

If A was the active drive instead of C, the formula of the command should have been the following:

```
A>cd c:\dos <RET>

A>_
```

The prompt shows that C has not automatically become the active drive. But issuing a DIR C: instruction proves that the sub-directory DOS has been activated in drive C. If we transfer a copy of a file in drive A to the hard disk, it will end up in the sub-directory in question.

In order to switch from drive A to drive C, we have to issue another instruction as well, either before or after the CD command, namely:

```
A>c: <RET>

C>_
```

As a result, all files in the sub-directory DOS are at our disposal.

To return to the level of the normal or root directory, we type:

```
C>cd \ <RET>

C>_
```

Now that we are back at our starting-point, we may use the opportunity to create a second sub-directory and name it DBASE. This one is used, for instance, for our dBASE files.

To achieve this we need the following command:

```
C>md \dbase <RET>

C>_
```

We have to bear in mind that we cannot rename a sub-directory once we have given it a name. Therefore, we should choose this name carefully.

We may give a sub-directory a set of sub-directories of its own. The same also applies to such a sub-sub-directory created in this way, and so on.

For example: suppose we want to extend the sub-directory named DBASE with a sub-sub-directory named MEMBERS. In that case we have to formulate the command as follows:

```
C>md \dbase\members <RET>
```

Note that we first type a backslash to mark the root directory, then the name of the directory of the next level, that is, in this case DBASE, and finally, preceded by a second backslash, the name of the sub-sub-directory, in this case MEMBERS. To check we type:

```
C>cd \dbase <RET>

C>dir <RET>

 Volume in drive C has no label
 Directory of  C:\dbase

.            <DIR>      12-21-88   2:48p
..           <DIR>      12-21-88   2:48p
MEMBERS      <DIR>      12-21-88   2:53p
        3 File(s)  21102406 bytes free
```

To switch to the level of the sub-directory we have just created, we type:

```
C>cd \dbase\members <RET>
```

Checking this again by means of a DIR command, we see:

```
 Volume in drive C has no label
 Directory of  C:\dbase\members

.            <DIR>      12-21-88   2:53p
..           <DIR>      12-21-88   2:53p
        2 File(s)  21102406 bytes free
```

The formula to return to the previous level is:

```
C>cd .. <RET>
```

The two dots are considered to symbolise the level in which the sub-directory was created.

To switch from one sub-directory to another, for instance DOS, we type, regardless of the active sub-directory at the moment of the instruction:

```
C>cd \dos <RET>
```

In order to return to the level of the root directory, we may restrict the command to the following, as we have already seen on page 86:

```
C>cd \ <RET>
```

If we want to check which sub-directory is the active one, we type:

```
C>cd <RET>
```

Assuming that the root directory is now active, the answer is:

```
C:\

C>_
```

And if the active sub-directory is DOS, for instance, the result is:

```
C:\dos

C>_
```

5.6: Displaying the map - TREE

No matter how the hard disk is arranged and which (sub-)directory is active, we can simply check the map, or rather the organisation of the directory. For this purpose, we use the TREE command.

Assuming that the TREE.COM file is in the root directory, we type the following two commands to start the TREE procedure:

```
C>path \ <RET>

C>tree <RET>
```

The result of the PATH command will be explained on page 90.

If the version of the operating system is 3.0 or higher, we only need to issue a single command, namely:

```
C>tree <RET>
```

If the TREE.COM file is not present in the root directory but in a sub-directory, for instance in DOS, we type:

```
C>path \dos <RET>

C>tree <RET>
```

In this case too, one single command will do if we have version 3.0 or higher. In this example we then type:

```
C>\dos\tree <RET>
```

In any case, this results in a survey like the following being displayed on the screen:

```
Directory Path Listing for Volume: No Name


Path: Root

Sub-Directories:  DOS
                  DBASE


Path: \DOS

Sub-Directories:  None


Path: \DBASE

Sub-Directories:  MEMBERS
```

And so on.

We may also add the parameter /f to a TREE command. Thus:

C>tree/f <RET>

As a result, we not only see a survey like the one above, but the contents of each separate sub-directory as well, for instance:

```
Directory Path Listing for Volume: No Name


Path: Root

Sub-Directories:  DOS
                  DBASE

Files:            ANSI    .SYS
                  APPEND  .COM
                  ASSIGN  .COM
                  ATTRIB  .EXE
                  CHKDSK  .COM
                  COMMAND .COM
                  COMP    .COM
                  DISKCOMP.COM
                  DISKCOPY.COM
                  DRIVER  .SYS
                  EDLIN   .COM
                  EXE2BIN .COM
                  FDISK   .COM
                  FIND    .EXE
                  FORMAT  .COM
                  GRAFTABL.COM
                  GRAPHICS.COM
```

And so on.

5.7: Creating a path

But what about the communication between the root directory and a sub-directory or between two sub-directories?

Assume we are working on the level of sub-directory DBASE, and that we want to operate a program that is in another sub-directory, for instance the system utility CHKDSK, in a sub-directory with the name DOS. The only way to activate a program in a different sub-directory is to describe this place very accurately. In this case the command should be:

C>\dos\chkdsk <RET>

So we have to make clear to the operating system that it has to search in the sub-directory DOS, while skipping the root directory, in order to start the program with the same name by means of the CHKDSK command. This is rather difficult. It would be easier if we could just give the command and if the system would find its way on its own. This is possible, provided that we create a path in advance, by means of a PATH instruction.

For example, if we want to instruct the system to find a program file in either the active sub-directory or the root directory, we create the path in the following manner:

C>path \ <RET>

And if we want to indicate the sub-directory DOS as a possible place where it can be found, we type the following command:

C>path \dos <RET>

We may also instruct the system to check both the root directory and a sub-directory, for instance DOS. To do so, we need only one PATH command. We use a semi-colon to separate the various routes in the formula:

C>path \;\dos <RET>

If we want to know whether a PATH command is already in operation, and if so which one, we use the basic command. Thus:

C>path <RET>

In this case, the system's answer is:

PATH=\;\DOS

If there is no PATH command in operation, we see:

PATH=\

Besides, a PATH command remains in operation as long as we do not turn off the computer, or do a RESET (see page 18), or issue another PATH command.

5.8: Disguising a sub-directory - SUBST

We sometimes have to deal with a program that dates back from a time when sub-directories did not yet exist. Such programs are very well capable of functioning on a hard disk, and in a sub-directory as well, but it is not able to trace and address a file in a different sub-directory.

Initially, the only alternative was to gather all necessary files in the same (sub-)directory, but the introduction of the PC-DOS and MS-DOS version 3.1 provides us with a more elegant solution to the problem, i.e. the SUBST procedure (for Substitute). It enables us to disguise a sub-directory as a drive. As a result, WordStar, or any other old-fashioned program, is fooled.

Suppose we have created a sub-directory named TEXTS on the hard disk, and a second named LETTERS. Assuming that the SUBST.EXE file is in the root directory, we now give the following commands if we want to disguise these two sub-directories as drive D and drive E, respectively:

```
C>subst d: c:\texts <RET>

C>subst e: c:\letters <RET>
```

There is no confirmation whatsoever, but if we call up the directory of 'drive D' or 'drive E', we are suitably informed.

To find out whether sub-directories have already been disguised as drives, and if so under which letter, we have to give the basic SUBST command:

```
C>subst <RET>
```

In this example the answer is:

```
D: => C:\TEXTS
E: => C:\LETTERS
```

In order to cancel a SUBST instruction, we type a SUBST command followed by the letter of the 'drive' in question, a space and the parameter /d (for delete). Thus:

```
C>subst d: /d <RET>

C>subst e: /d <RET>
```

Disguising a sub-directory is also useful when we are working with a complicated structure of three or more (sub-)directories. In that case it saves us quite some typing if we can replace this long description of routes by a letter and a colon.

In the normal ('default') state of the operating system, the number of drives (both real and artificial) can amount to no more than five, so the letter E is the last one. How we can increase this number will be explained on page 127.

5.9: Duplicating (2) - BACKUP, RESTORE

Should there ever be something wrong with the hard disk, or if we accidentally start a FORMAT procedure in spite of all precautions, then this will show how sensible it was to copy the contents regularly.

But again, the storage capacity is causing a problem. Not only does it take some sixty floppy disks to back up one 20 Mb hard disk completely, but also is it time-consuming to copy all files of each directory one by one. Plus, the size of a file may exceed the storage capacity of a floppy disk.

The most simple solution would be a second hard disk or a tape streamer. Indeed, a number of users and manufacturers resort to such facilities, but they are quite expensive.

Less simple is the method IBM and Microsoft offer us via the operating system. It does take a large number of floppy disks but the copy procedure itself is carried out more or less automatically.

For this purpose we type the following formula, assuming that a file named BACKUP.COM or BACKUP.EXE can be reached on drive C, and that drive A is available for the backup procedure:

```
C>backup c:\*.* a:/s <RET>
```

The source (in this example c:*.*) and the destination (in this example a:) must always be explicitly mentioned. The parameter /s tells the BACKUP program that all subdirectories of the root directory must be copied too.

Next, we see a message like the following on the screen:

```
Insert backup diskette 01 in drive A:
Warning! Diskette files will be erased
Strike any key when ready
```

We must then insert a blank, formatted disk in drive A and strike a key. As soon as we do that, we will see:

```
***  Backing up files to diskette 01  ***
```

File after file and directory after directory will be copied, until the disk is filled to the rim. Then the following message will be displayed:

```
Insert backup diskette 02 in drive A:
Warning! Diskette files will be erased
Strike any key when ready
```

We exchange the filled disk for an empty one and strike a key.

```
***  Backing up files to diskette 02  ***
```

We continue until a message tells us that the procedure is completed:

```
Successful backup
```

In this procedure, we may also make a selection. For instance, should we just want to make a backup copy of the files in the sub-directory LETTERS, the correct formula is:

C>backup c:\letters*.* a: <RET>

We may even narrow the procedure down to a certain group of files. If we want to limit the procedure to the files with the extension .TXT in the sub-directory NOTES, we type the following formula:

C>backup c:\notes*.txt a: <RET>

If we add the parameter /m (for modified), the procedure is limited to the files that have been created or altered since the previous BACKUP procedure - provided we use the same set of disks. For instance:

C>backup c:\notes*.txt a:/m <RET>

If we use the parameter /d (for date), followed by a colon and a date, only the files that have been created and/or altered on or since that date will be copied. For instance, if we want to copy all files in all directories created on or after 20 December 1988, we type the following command:

C>backup c:*.* a:/s/d:12-20-88 <RET>

A disadvantage of the BACKUP procedure is that we cannot use the copied files in a normal way - neither to inspect the contents, nor to alter them.

Should it ever be necessary to restore the contents of the hard disk by means of our collection of backup copies, hopefully not made too long ago, we have to start a reverse procedure.

First, we type, assuming we have file named RESTORE.COM or RESTORE.EXE available on a diskette in drive A, the following command:

A>restore a: c:/s <RET>

As a result, we see:

```
Insert backup diskette 01 in drive A:
Strike any key when ready
```

Then we must take out the diskette in drive A, insert the first diskette of the backup set in drive A and strike a key. If all goes well, we will see:

```
*** Restoring files from diskette 01 ***
```

And so on, until the procedure is completed:

```
Normal completion
```

If we have to restore just one single file, for instance TEST, and the destination is the sub-directory NOTES, the correct formula is:

A>restore a: c:\notes\test <RET>

6: WORKING WITH EDLIN

Every operating system is equipped with a facility that enables the user to create, store or revise a text file. MS-DOS and PC-DOS are no exception. All versions are supplied with a utility called EDLIN.COM, which is suitable for making notes and storing them in a file, composing a letter or writing and editing a so-called batch program.

However, EDLIN is less suitable to be used as a word processor, in other words, as a program to write more extensive items such as reports and manuscripts. Compared with real word processing programs such as WordStar, WordPerfect and Microsoft Word, EDLIN is too cumbersome.

On the other hand, EDLIN is automatically supplied with the system, and it is quite instructive to learn how it works.

The main difference between EDLIN and WordStar, WordPerfect and similar programs is that the latter are so-called screen editors. They are programs that enable us to move the cursor over the entire surface of the screen - that is, a total of 25 lines, each of which can contain 80 characters - and which can carry out a certain instruction on any of these 2000 positions.

EDLIN is merely a line editor, as the name more or less indicates. It is a program in which the range of the cursor is restricted to one line, and we can only carry out instructions on the positions on that particular line. This may sound more complicated than it in fact is, because in view of the experiences we had with the computer so far - i.e. typing and issuing commands, creating a text file with the COPY CON: method (see page 50) - the range of the cursor was also restricted to one line.

In addition, EDLIN has the possibility of displaying the lines that have already been passed on to the computer and stored in a file, after which we can erase, alter or replace them again, if we want to, without affecting the rest of the contents of the file.

In order to facilitate this possibility, EDLIN is equipped with a number of tools. In the first place, they include a number of built-in commands consisting of one single letter. We can use these commands to pass on our instructions to the program.

The second facility is the automatic numbering of lines in every file we create in EDLIN. This enables us to indicate accurately which line we wish to view or correct on the screen.

In the third place, we may use a number of special keys that perform a specific function which facilitate operations often used, such as recalling lines and making corrections. They are the function keys we became acquainted with on page 22.

In order to manipulate a file any further, for instance to rename, erase or send it to an input or output channel, we have to use one of the system facilities that we became acquainted with in the previous chapters. What we have to do to print a file will be discussed on pages 122 and 174.

Let's make a start and, by way of experiment, create a text file named MEMO with the same contents as the TEXT file on page 50.

Assuming that the EDLIN.COM file is in the root directory of the hard disk (drive C), and that drive A contains a disk with a sufficiently available storage capacity, the command to start creating a file is:

```
C>edlin a:memo <RET>
```

This results in the following message on the screen:

```
New File
*_
```

The asterisk is EDLIN's prompt - that is, the invitation to indicate what we want EDLIN to do.

To open a new file we type an 'i' (or an 'I'):

```
*i <RET>
     1:*_
```

As a result, we see the signal that we can start entering our text.

Initially we proceed as we did when using the COPY CON: procedure (see page 50). We act as if we were sitting at a typewriter: we strike the RETURN key at the end of each line. Thus:

```
*i
     1:*It is advisable to enter the date and_
```

The maximum number of characters permitted per line is 128 and the maximum number of lines is (in theory) 65,535.

```
*i
     1:*It is advisable to enter the date and
     2:*time when loading the system, so that
     3:*_
```

We can remove a typing error with the BACKSPACE key or the instruction ^H (see page 18).

Instead of erasing all the characters of a line one by one, we may erase the whole line by striking the ESC key. A backslash (\) then appears at the end of the line, to indicate that the line will not be entered into the computer. The cursor reappears at the start of the next line.

To abandon our writing for any reason, we type ^C or ^<BREAK>.

As soon as we are finished, we hit the F6 key, or we give the instruction ^Z, in both cases followed by striking the RETURN key.

Finally, the screen should show the following:

```
C>edlin a:memo
New File
*i
     1:*It is advisable to enter the date and
     2:*time when loading the system, so that
     3:*we later can check when a file like this
     4:*was created.
     5:*^Z
*_
```

6.1: Closing - E or Q

There are three ways of concluding our activities under EDLIN.

We employ the first method when we have created a new file and we want to store it on a disk, or when we have altered a file and we want to save the updated version. In those cases we type an 'e' (for 'end') immediately after the prompt, and we hit the RETURN key:

```
*e <RET>

C>_
```

With a DIR command we can check that our MEMO file has indeed been saved:

```
C>dir a: <RET>

 Volume in drive A has no label
 Directory of  A:\

MEMO                 134  12-24-88   1:19p
        1 File(s)    361472 bytes free
```

The second method can be applied if we have no desire to save our text. In this case we type a 'q' (for 'quit'):

```
*q <RET>
```

Instead of the system prompt we first see a query as to whether we really want to throw away this version of our file:

```
Abort edit (Y/N)? _
```

We have to type a 'y' to get the system prompt back on the screen. By means of a DIR command we can check whether or not the file has been saved.

The third way to stop is a RESET procedure (see page 18). However, this should only be used in emergency cases.

Suppose that a RESET procedure was unavoidable. Then the directory of the disk in drive B shows us that an entry had been reserved for our file:

```
C>dir a: <RET>

 Volume in drive A has no label
 Directory of  A:

MEMO     $$$         0  12-24-88    1:25p
        1 File(s)    362496 bytes free
```

Note that the extension to the file name is $$$ and that the size of the file is 0 bytes. That means, the file is empty.

6.2: Re-opening

We may want to take another look at our MEMO file and edit it. If so, we need to re-open it. Just as when we were creating it, we type:

```
C>edlin a:memo <RET>
```

There are three possible ways in which EDLIN can respond. The first is:

```
New file
*_
```

In this case we have apparently made a typing error. EDLIN assumes that we want to open a new file.

We solve the problem by typing a 'q' as soon as the prompt appears. After that, we can try again.

A second possible response of EDLIN is just the prompt, and the absence of any message at all:

```
*_
```

In that case the file is too large for the memory space available. On page 116 we shall see how to find a way out of this situation.

A third possible reaction is:

```
End of input file
*_
```

This is the one we need. This message means that the requested file has been loaded into the computer's memory, and that we may now give our next command. If we type an 'i', as we did when creating the file (see page 96), we see:

```
*i <RET>
     1:*_
```

We get the idea that the first line of the file is empty. It is not. The cause of the apparent confusion is the real nature of the 'i' command, which is short for 'insert'. On page 100 we will go into more detail about this command and how to use it.

If we have no intention of inserting anything, we should type a ^C or a ^<BREAK> instruction. The prompt will reappear on the next line:

```
     1:*^C
*_
```

This gives us another chance to get EDLIN to do what we had in mind.

6.3: The .BAK file

Suppose we have created the file MEMO, stored it on the disk, re-opened it and saved it once more, after which EDLIN is replaced by the regime of the operating system.

To see what the results of these operations are, we give a DIR command:

```
C>dir a: <RET>

 Volume in drive A has no label
 Directory of  A:

MEMO     BAK       134  12-24-88   1:19p
MEMO               134  12-24-88   1:40p
        3 File(s)     360448 bytes free
```

We see that in addition to the (revised) MEMO file, there is now a second file named MEMO, but with the extension .BAK. This is an abbreviation of Backup file. This .BAK file contains the previous version of the MEMO file - that is, the way the file was before we opened and saved it a second time.

We need not to go into technical detail of what the computer does, but it comes down to the following: as soon as we re-open a file, EDLIN loads part of what was on the disk into memory. This will form the basis for the new version. Meanwhile, the original version of the file stays on the disk, without any of the corrections or alterations we may be adding. This version is given the extension .BAK at the very moment we ask EDLIN to store the new, updated version.

Thus, if anything goes irrevocably wrong with the file in question, we can always fall back on this previous version. But first we have to type a REN command (see page 52) to give this .BAK file another name, or at least a name without a .BAK extension, because EDLIN will not touch a .BAK file.

While the knowledge that EDLIN automatically keeps a reserve file is very reassuring, we have to make sure that the size of the file we are working on, plus its .BAK version, plus some working space, stays well within the amount of storage room available on the disk.

As a rule of thumb, the file should not be larger than half the storage room that is available. It is better to restrict the size to a third. This leaves room for the new .BAK file plus the previous one, if any, which is erased only after the updated version of the file has been stored.

In any case, we are well advised to keep track of the available storage room on a disk. In version 1 of the system, we issue a DIR command (see page 28) to find out about the file's size, and a CHKDSK command (see page 32) to stay informed about the remaining storage room. In versions 2 and 3 of the system, we can just issue a DIR command.

Apart from that, we should always keep a copy of our file on another disk. Anything can happen. The original disk may get lost, for instance, or may even get damaged by an upset cup of tea.

It is a good habit therefore to make a copy of the disk regularly, using one of the system's copy methods (see pages 34 and 55).

6.4: Inserting - I

If we want to extend a file by adding a number of lines, we need an 'i' (for 'insert') after re-opening the file. The precise formulation of the command, however, will depend on the place where we want to add the new lines of text.

Suppose that we want to add two lines to the start of our MEMO file, for instance, a title and a blank line. We open the procedure as follows:

```
C>edlin a:memo <RET>
End of input file
*i <RET>
        1:*_
```

Now we have the opportunity to type in a new number 1 line:

```
        1:*About the marking of files <RET>
        2:*_
```

To leave the second line blank, we need only hit the RETURN key:

```
        1:*About the marking of files
        2:* <RET>
        3:*_
```

If we want to leave it at that, we just give a ^C or ^<BREAK> instruction:

```
        1:*About the marking of files
        2:*
        3:*^C
*_
```

To see the results of our manipulations, we can type an 'l', for 'list', followed by striking the RETURN key (see page 102). Thus:

```
*l <RET>
        1: About the marking of files
        2:
        3:*It is advisable to enter the date and
        4: time when loading the system, so that
        5: we later can check when a file like this
        6: was created.
*_
```

It is quite clear what has happened: the lines 1 and 2 have been added at the beginning, pushing the original lines down. What had been lines 1 and 2 are now lines 3 and 4.

Note that an asterisk is shown at the beginning of line 3. This is EDLIN's way of indicating that line 3 is currently selected.

We now have a choice. We can either save our new version with the added title, in which case we should hit the 'e' and the RETURN key, or we can tell EDLIN to forget it by typing a 'q' and striking the RETURN key. As we have seen on page 127, we will be asked to confirm our decision:

```
Abort edit (Y/N)? _
```

We have to type a 'y' to confirm our decision. If we type an 'n', however, the EDLIN prompt reappears.

Another option is to carry on inserting text.

Suppose we want to add a few lines at the end of the file. In that case we should precede the 'i' by the number of the line where we want our text to be inserted.

Because our file ends after the last additions on line 6, we now want to start the new text on line 7. Thus:

```
*7i <RET>
     7:*_
```

We type the lines we want to add:

```
     7:*   Especially when we have more than one
     8:*version of a file and there is a copy of
     9:*each version, this procedure can save us
    10:*a lot of time and effort.
    11:*_
```

As soon as we have finished, we give a ^C or ^<BREAK> instruction:

```
    11:*^C
*_
```

To see what our file now looks like, we type an 'l' again and strike the RETURN key. We then see:

```
*l <RET>
     1: About the marking of files
     2:
     3: It is advisable to enter the date and
     4: time when loading the system, so that
     5: we later can check when a file like this
     6: was created.
     7:    Especially when we have more than one
     8: version of a file and there is a copy of
     9: each version, this procedure can save us
    10: a lot of time and effort.
*_
```

Note that none of the lines is prefixed by an asterisk. This is because line 11 is now selected, but has not yet been used.

Once again, we have a choice between saving our changes with an 'e' or aborting them by typing a 'q'.

6.5: Reproducing - L or P

It will often be necessary to inspect the contents of an EDLIN file. If we are under the control of the system, we can use the TYPE command from the system prompt, as we saw on page 51. But if we are in EDLIN, this command is not available. Then we can use the instruction 'l' (for 'list'). Thus:

```
*l <RET>
```

Immediately the contents of the file will fill the screen.

In cases where the size of the file is larger than 23 lines, only the first 23 lines will be displayed - exactly a screenful. For instance:

```
*l
         1:*About the marking of files
         2:
         3: It is advisable to enter the date and
         4: time when loading the system, so that
         5: we later can check when a file like this
         6: was created.
         7:    Especially when we have more than one
         8: version of a file and there is a copy of
         9: each version, this procedure can save us
        10: a lot of time and effort.
        11:    Some computer makes are fitted with
        12: a built-in clock and calendar. A battery
        13: ensures that they keep ticking when the
        14: computer is switched off. The operating
        15: system automatically loads the time and
        16: date from the clock and the calendar, so
        17: we don't have to type those in.
        18:    A computer not supplied with a built-
        19: in clock can have one fitted later. We
        20: then have to buy an expansion card. This
        21: card will usually also provide us with
        22: other facilities, such as extra memory,
        23: or extra input and output channels. Our
*_
```

In case the active line is not line 1, but for instance line 17, we would see this line together with the eleven preceding and the eleven following lines (in other words, lines 6 to 28, or up to the end of the file, whichever comes first).

To see another screen, we should type the next line number, followed by an 'l'. For instance:

```
*24l <RET>
        24: dealer may be able to give more details.
*_
```

If we do not want to see a file from line 1 but, for instance, from line 10, we just type the line number followed by an 'l':

```
*10l <RET>
```

As a result, we see the line we mentioned, followed by the next 22 lines.

To restrict the display to a limited number of lines, for instance to the lines 13 to 22, we type both numbers, separated by a comma and followed by an 'l'. Thus:

```
*13,22l <RET>
     13: ensures that they keep ticking when the
     14: computer is switched off. The operating
     15: system automatically loads the time and
     16: date from the clock and the calendar, so
     17: we don't have to type those in.
     18:    A computer not supplied with a built-
     19: in clock can have one fitted later. We
     20: then have to buy an expansion card. This
     21: card will usually also provide us with
     22: other facilities, such as extra memory,
*_
```

If we want to see the file from the beginning up to some particular line, for instance line 9, we type just the comma, followed by the number of the last line requested plus an 'l':

```
*,9l <RET>
      1:*About the marking of files
      2:
      3: It is advisable to enter the date and
      4: time when loading the system, so that
      5: we later can check when a file like this
      6: was created.
      7:    Especially when we have more than one
      8: version of a file and there is a copy of
      9: each version, this procedure can save us
*_
```

EDLIN is also equipped with the command 'p' (for 'page'). It produces the same kind of results as an 'l', except for the position of the asterisk.

Using the 'p' command automatically moves the asterisk to the last line specified. For example:

```
*11,17p <RET>
     11:    Some computer makes are fitted with
     12: a built-in clock and calendar. A battery
     13: ensures that they keep ticking when the
     14: computer is switched off. The operating
     15: system automatically loads the time and
     16: date from the clock and the calendar, so
     17:*we don't have to type those in.
*_
```

6.6: Editing keys

Under the regime of EDLIN, the function keys work in the same way as under the regime of the system itself (see page 22). Thus, we can use the F3 key to bring back the entire last-entered command line, or the F1 to bring it back character by character, etc.

In addition, there are a number of other keys that can be described as editing keys, and that also work both in EDLIN and in the system itself.

One example is the key marked INS, for Insert. Its task is to add one or more characters on the command or text line which is in the buffer at that time. Suppose we type the command to edit our MEMO file and forget to name A as the source drive. Thus:

```
C>edlin memo <RET>
```

The result is that EDLIN will look in vain on drive C (the hard disk) for the file, and then assume that a new file is required. Hence its response on the screen will be:

```
New file
*_
```

We have to strike the 'q' and RETURN keys, followed by a 'y', to get the system prompt back on the screen.

Our original command to open the file is still in the buffer, however, so if we hit the F2 key and then type a 'm', we see:

```
C>edlin _
```

After striking the INS key, we type in the forgotten part of the command:

```
C>edlin a:_
```

Tapping the F3 key after that gives us the following:

```
C>edlin a:memo_
```

Hitting the RETURN key is then enough to pass on the modified command to the system.

Close to the INS key we find a key marked DEL, for Delete. This key has the opposite effect. Its task is to delete a character or a series of characters in a command or a text line that is stored in the buffer.

Suppose once again that we want to edit our MEMO file with EDLIN. This time the file is on the hard disk. However, we accidentally mention 'a' as the drive:

```
C>edlin a:memo <RET>
```

EDLIN will look for the file on the disk in drive A. Not finding it there, it assumes that a new file is required and responds:

```
New file
*_
```

Again, we have to hit the 'q' and RETURN keys, followed by a 'y', to get the system prompt back on the screen.

The command is still in the buffer. Therefore, if we strike the F2 key and type an 'a', we will see:

```
C>edlin _
```

Using the DEL key we can now erase the unwanted part of the command. We hit this key twice and then we hit the F3 key. The result will be:

```
C>edlin memo_
```

Once again hitting the RETURN key is sufficient to pass on the modified command to the system.

Some keyboards are equipped with a group of four keys with arrows. This group is usually on the right side of the keyboard, in (or next to) the numeric keypad. If there is a group of keys like this, the backspace key, that is, the one with an arrow pointing to the left (not to be confused with the RETURN key), has the same function as the ^H instruction (see page 18). It serves to delete the character to the left of the cursor.

The arrow pointing to the right has the same function as the F1 key. As long as we hold it down, the contents of the buffer are printed on the screen, one character at a time.

Usually we find on the left side of the keyboard a key marked with the term TAB or with two arrows pointing in opposite directions. This key will move the cursor a fixed number of places (usually eight) to the right:

```
C>edlin memo_
```

This can be useful when typing tables, or writing programs in machine code.

Another key that we may use in editing is the one marked ESC, for Escape. We press it if we want EDLIN to ignore a command or a text line. For instance:

```
C>edlin memo\

_
```

Note that the backslash follows the last character and that the cursor has moved to the next line. We can easily find out whether the contents of the buffer are still intact.

Computers operating under the control of PC-DOS or MS-DOS usually have a number of additional editing keys marked HOME, END, PG UP and PG DN, apart from the arrow keys. These have no function under EDLIN. They can only be used with programs like WordStar, which use the whole screen as a workshop.

6.7: The active line

Regardless of the phase we are in using EDLIN, there is always one active line. When we open or re-open a file, for instance, the first line is the active one. We recognise this line by the asterisk between the number and the text. Of course, we should not confuse this with the EDLIN prompt.

Suppose we just have re-opened our MEMO file. Striking the RETURN key is enough to make the next line the active line:

```
C>edlin a:memo
End of input file
* <RET>
     2:*
     2:*_
```

If we hit the RETURN key again, the prompt reappears. This enables us to browse rapidly through the file:

```
     2:*
     2:* <RET>
* <RET>
     3:*It is advisable to enter the date and
     3:* <RET>
* <RET>
     4:*time when loading the system, so that
     4:* <RET>
*_
```

And so on. If we want to work with the active line and we cannot remember its number, we only need type a full stop. For example:

```
*. <RET>
     4:*time when loading the system, so that
     4:*_
```

And if we want to add a line at the end of the file and cannot remember the number of the last line, we should use the # symbol, followed by the 'i' instruction. The active line now follows right after the last line of the file. For instance:

```
*#i <RET>
    24:*_
```

We also have the opportunity to use plus (+) and minus (-) signs to indicate the position of a line relative to the active line.

For instance, if we want to use an 'l' command (see page 102) to show the thirteen lines preceding the active line, we type:

```
*-131 <RET>
```

6.8: Editing

It is when we want to make corrections that we notice just how cumbersome working with EDLIN can be. Instead of seeing a screen filled with text, where we can navigate at will with the cursor, we are forced to plod through a file a line at a time.

Therefore, it is a good habit to have a hard copy (that is, a printed copy) of a file at hand, and then mark the lines to be altered before starting on the editing work.

Suppose that we want to make an alteration in line 20 of our MEMO file. We start off by loading the file. As soon as the EDLIN prompt appears, we can type the wanted line number. The visible result will be:

```
C>edlin a:memo
End of input file
*20
     20:*then have to buy an expansion card. This
     20:*_
```

We may now formulate the text of the line in the way we want it.

As soon as we are ready, we strike the RETURN key and we will see the EDLIN prompt back on the screen. We can then ask for another line to edit, issue another command, or do whatever we wish.

If we have called up a specific line and we decide on second thoughts to make no corrections in it, we can type a ^C or ^<BREAK> instruction, which will cause the prompt to reappear immediately:

```
C>edlin a:memo
End of input file
*20
     20:*then have to buy an expansion card. This
     20:*^C
*_
```

To save a correction we have made and then to return to the control of the system, we must type an 'e'.

If, on the other hand, we would prefer to scrap the alterations we have made, we should type a 'q'. In this case, just as when we opened the file (see page 96), we first have to confirm our intended course of action:

```
Abort edit (Y/N)? _
```

We need to type a 'y' to have the system prompt reappear.

In the editing process, the function keys F1 to F4 and some editing keys are essential. On pages 22 and 23 we have seen how we can make use of them under the control of the operating system itself. The next four pages will describe how we can make effective use of them in EDLIN.

Let's now try and find out how the function and editing keys work under the control of EDLIN. Again we call up line 20 of our MEMO file:

```
C>edlin a:memo
End of input file
*20
     20:*then have to buy an expansion card. This
     20:*_
```

We can use the F3 key to print out the whole line on the screen:

```
20:*then have to buy an expansion card. This
20:*then have to buy an expansion card. This_
```

There is little point in doing this if nothing has been changed in the line. Therefore, we use the F3 key mainly in combination with another key.

If we strike the F1 key, or the key with the arrow pointing to the right, instead of F3, we see:

```
20:*then have to buy an expansion card. This
20:*t_
```

As we strike F1 again, EDLIN projects the next letter in the line, and so on. The moment that the cursor arrives at a wrong character, we can type another in its place. Then we can use F1 or F3 to complete the line.

We can use the F2 key to reproduce a text line up to (but not including) a specific character.

For instance, if we want to reproduce the part of the line up to 'expansion', we should first hit the F2 key and after that an 'x'. The result is:

```
20:*then have to buy an expansion card. This
20:*then have to buy an e_
```

The F4 key can be used to reproduce part of a line from a certain point.

Suppose we only want to see the section of the line starting with the word 'buy'. In that case we first strike the F4 key, then we type a 'b', and finally we hit the F3 key. As a result, we see:

```
20:*then have to buy an expansion card. This
20:*buy an expansion card. This_
```

The F5 key serves to erase (part of) a line from the buffer, or to replace it with another line, without having to use the RETURN key. In that way we can edit the line to measure. If for instance, we use the F2 key followed by a 'c' and then the F5 key, we see:

```
20:*then have to buy an expansion card. This
20:*then have to buy an expansion @

   _
```

The '@' means that the rest of the line has been scrapped.

Suppose we want to add something to the line, for instance, between 'buy' and 'an'. In that case we have to take quite a few steps.
First of all we have to call up the line:

```
*20 <RET>
     20:*then have to buy an expansion card. This
     20:*_
```

The first words of the sentence can stay, so we can either hit the F1 key or the arrow pointing to the right sixteen times. The result is:

```
20:*then have to buy an expansion card. This
20:*then have to buy _
```

It would have been easier to hit the F2 key, followed by an 'a', with the result:

```
20:*then have to buy an expansion card. This
20:*then h_
```

Note that the cursor was stopped by an earlier 'a'. By hitting the same two keys again, we would have reached the next 'a' on the line:

```
20:*then have to buy an expansion card. This
20:*then have to buy _
```

If we want to add something to the line, we first have to strike the INS key, to prevent the new text from erasing the existing one at the same place. After that, we can type in our additional text:

```
20:*then have to buy an expansion card. This
20:*then have to buy an accessory known as _
```

Tapping the F3 key will automatically add the rest of the line:

```
     20:*then have to buy an expansion card. This
     20:*then have to buy an accessory known as
an expansion card. This_
```

Note that part of the text has now spilled over to the next line.
The reason is that the screen can take a maximum of only 80 (or 40, depending on the setting) characters per line (see page 120). Superfluous characters will move to the next line, but EDLIN still regards the whole as one line.

The next steps involve moving the section of the line from the word 'as' to an extra line, which we want to insert between lines 20 and 21.
We should first hit the RETURN key, to pass on the new line to EDLIN and get the prompt back on the far left. Thus:

```
     20:*then have to buy an expansion card. This
     20:*then have to buy an accessory known as
an expansion card. This
*_
```

Next, we have to type the number 21 and an 'i', which is the command to insert a new line 21 (see page 100):

```
*21i <RET>
    21:*_
```

By tapping the F3 key we check that the revised line 20 is still in the buffer.

As we do not need a whole line, we first strike the ESC key to tell EDLIN that we do not want the line in this form in memory.

A backslash will appear at the end of the line, to indicate that EDLIN will ignore the text. The cursor will move down a line, but the contents of the buffer will not be touched:

```
*21i
    21:*then have to buy an expansion card. This
    21:*then have to buy an accessory known as
an expansion card. This\
```

To scrap the line up to and including the word 'known', we may use the DEL key, which removes one character from the buffer (in a forward direction) every time we hit it. However, in this example we would have to use it some forty times.

Generally, a better way is to hit the F4 key, type an 'a', strike the F3 key, and finally use the F5 key to send the result to the buffer.

Because there is more than one 'a' in the line, we will have to repeat the procedure with F4, 'a' and F3, before using the F5 key each time to send the new result to the buffer:

```
* 21i
    21:*then have to buy an expansion card. This
    21:*then have to buy an accessory known as
an expansion card. This\
        ave to buy an accessory known as an expa
nsion card. This@

        _
```

In the end, the screen will appear like this:

```
* 21i
    21:*then have to buy an expansion card. This
    21:*then have to buy an accessory known as
an expansion card. This\
        ave to buy an accessory known as an expa
nsion card. This@
        an accessory known as an expansion card.
This@
        accessory known as an expansion card. Th
is@
        as an expansion card. This@

        _
```

And there we are.

If we now strike the RETURN key, EDLIN will read this new line as line 21 and return with the prompt for line 22:

```
22:*_
```

We do not yet have to worry about line 22, so we give a ^C or a ^<BREAK> instruction to end the procedure. The EDLIN prompt will reappear.

To view lines 20 to 24 in their new form, we type:

```
*20,241 <RET>
```

We see the following:

```
    20: then have to buy an accessory known as
an expansion card. This
    21: as an expansion card. This
    22:*card will usually also provide us with
    23: other facilities, such as extra memory,
    24: or extra input and output channels. Our
*_
```

Obviously, we still need to clean up line 20, so we load that line again:

```
*20 <RET>
    20:*then have to buy an accessory known as
an expansion card.
    20:*_
```

By using the F2 key and the 'a' key four times (more or less the opposite of what we did with the F4 key and the 'a' on the previous page), we will produce the following result:

```
20:*then have to buy an accessory known _
```

If we hit the RETURN key, the revised line is put into the buffer. EDLIN's prompt reappears on the screen.

If we give another 'l' command, we see:

```
*20,241 <RET>
    20:*then have to buy an accessory known
    21: as an expansion card. This
    22: card will usually also provide us with
    23: other facilities, such as extra memory,
    24: or extra input and output channels. Our
*_
```

Now the lines from 21 on down remain to be reorganised.

We can, of course, use the same procedure to make the lines to measure, but it is quite a lot of work. A simpler way is to type in the words we need to reach the required length from the line that follows and to tap the RETURN key. After that we can load the following line, erase the same words, append words from the next line, strike the RETURN key, and so on.

Cumbersome is just the right word to describe this process.

6.9: Searching - S

A useful feature of EDLIN is its find procedure. It enables us to search for a specific combination of characters (for instance a word, a number, a phrase or even a series of spaces) in a file.

Suppose we have just re-opened the MEMO file. To find out if the word 'battery' is present in the file, we use the command 's' (for 'search'):

```
*sbattery <RET>
      12:*a built-in clock and calendar. A battery
*_
```

As a result, EDLIN shows us the first location of the word.

Note that the command consists of a solitary 's', immediately followed by whatever we want found.

We should be careful not to include a space between the 's' and the characters, since that would order EDLIN to search for a space, followed by the word in question.

The exact formulation of the command is saved in the buffer. Therefore, to get EDLIN to look if the character sequence can be found elsewhere in the file, we need only type another 's':

```
*s <RET>
```

An 's' command is effective from the active line to the end of the file. As soon as the sequence of characters is found, the procedure stops. The line in which the characters have been found becomes the active line.

As with the 'l' command (see page 102), we can, if we wish, restrict the range of the search command to a specific part of the file, for instance from line 15 to line 25:

```
*15,25sbattery <RET>
```

We may instruct EDLIN to repeat a search procedure, until asked to stop. In that case the 's' should be preceded by a question mark. For instance:

```
*?sbattery <RET>
```

If the characters are found, they will be presented as follows:

```
      12: a built-in clock and calendar. A battery
O.K.? _
```

If we answer a 'y', the search is ended and the prompt will reappear on the screen. If we type an 'n', the procedure is repeated until EDLIN finds the next occurrence or gets to the end of the file.

6.10: Searching and replacing - R

In addition to the search procedure mentioned on the previous page, EDLIN is equipped with a facility which enables us to find specific characters or words, or whatever, and replace them by others. The command consists of the letter 'r' (for 'replace').

Suppose we want to replace the words 'an expansion card' in our MEMO file by 'a clock card'. In that case we give the following command:

```
*ran expansion^Za clock <RET>
```

Note that the division between the characters to be searched for and those to replace them is marked by the instruction ^Z. We can either use just the F6 key, or type the control instruction ^Z.

As a result of this command, EDLIN rapidly scans the file, and it shows us the first instance, if any, after making the required change. Thus:

```
          21:*then have to buy a clock card. This
*_
```

As with the 's' command, the exact formulation of the command stays in the buffer. Therefore, to get EDLIN to look if the character sequence can be found further on in the file again, we need only type another 'r'.

If EDLIN is unable to locate and replace the sequence, we will see on the screen:

```
Not found
*_
```

Just as with the find command 's', the replace command can be preceded by numbers, separated by a comma, to indicate the lines that the search and replace command refers to. For example:

```
*18,23ran expansion^Za clock <RET>
```

We may also add a question mark to the formula if we want the opportunity to choose whether or not to make the change:

```
*18,23?ra clock^Zan expansion <RET>
          21:*then have to buy an expansion card. This
O.K.? _
```

If we type 'y', EDLIN makes the correction final.

If we type an 'n', however, the line will be restored in its original form, and the program will seek the next occurrence of the expression, if there is one, and display it, again with the exchange made:

```
O.K.? n <RET>
*_
```

6.11: Erasing - D

The counterpart of the 'i' (for 'insert') command (see page 100) is 'd' (for 'delete'), which we use to remove a line of text.

Suppose we want to remove the blank line we inserted as line 2 (see page 100). We just type the line number, followed by a 'd':

```
C>edlin a:memo <RET>
End of input file
*2d <RET>
*_
```

We will not see any confirmation on the screen, just the EDLIN prompt.

To check whether anything has happened, we have to give an 'l' command (see page 102). And indeed, line 2 has been removed. Line 3 has taken its place and the rest of the text has also moved up one line.

Should we want to erase the active line, we need only type a 'd' on its own. In this case it is line number 1. Thus:

```
*d <RET>
*_
```

By giving an 'l' command, we get the file in its present form on the screen:

```
*l <RET>
      1:*It is advisable to enter the date and
      2: time when loading the system, so that
      3: we later can check when a file like this
      4: was created.
      5:    Especially when we have more than one
      6: version of a file and there is a copy of
      7: each version, this procedure can save us
      8: a lot of time and effort.
      9:    Some computer makes are fitted with
     10: a built-in clock and calendar. A battery
     11: ensures that they keep ticking when the
     12: computer is switched off. The operating
     13: system automatically loads the time and
     14: date from the clock and the calendar, so
     15: we don't have to type those in.
     16:    A computer not supplied with a built-
     17: in clock can have one fitted later. We
     18: then have to buy an accessory known
     19: as a clock card. This
     20: card will usually also provide us with
     21: other facilities, such as extra memory,
     22: or extra input and output channels. Our
     23: dealer may be able to give more details.
*_
```

If we wish, we can delete more than one line at a time. To do so, we have to type the first and the last line numbers of the block, separated by a comma, and followed by a 'd':

```
*5,8d <RET>
*_
```

We see that lines 5 to 8 have been removed from the file:

```
*l <RET>
     1:*It is advisable to enter the date and
     2: time when loading the system, so that
     3: we later can check when a file like this
     4: was created.
     5:    Some computer makes are fitted with
     6: a built-in clock and calendar. A battery
     7: ensures that they keep ticking when the
     8: computer is switched off. The operating
```

By omitting the first number, and only typing a comma followed by the number of the last line to be taken out, we get EDLIN to delete the lines from the active line up to and including the line that we mentioned:

```
*,7 <RET>
*_
```

At this point we still have the chance to decide whether or not we want to store the revised file; that is, without the lines that were deleted. If we do, we must type an 'e' from the EDLIN prompt:

```
*e <RET>
```

To undo the deletions we should type a 'q'. Before EDLIN scraps our changes, it gives us still another opportunity to think twice:

```
Abort edit (Y/N)? _
```

Thus, in case we feel we have taken the wrong path, there are several ways to retrace our steps.

Even if we have stored the new version of our MEMO file, we have a way to reconstruct its original form, since we still have the .BAK version, which was stored on disk before the latest changes were made (see page 99).

We cannot, however, edit a .BAK file directly. If we try, we get to see an error message:

```
Cannot edit .BAK file--rename file
```

The safest procedure is to make a copy of the .BAK file, at the same time giving it a new name, or at least another extension, for instance RESERVE. We have learned on page 56 how to do it. Now EDLIN accepts the file:

```
C>edlin a:reserve <RET>
   End of input file
   *_
```

6.12: Writing away - W

As mentioned on page 96, there may be times that we want to edit a file and do not see the 'End of input file' message on the screen.

We can conclude from this that the file in question was too long to fit in the memory available. In such a situation, we can use a 'w' (for 'write') to send part of the file, from line 1 up to and including the line number that we indicate, back to the disk. Let's try this with the MEMO file:

```
*21w <RET>
*_
```

At first sight, nothing has happened. But if we use an 'l' command to view what is left of the file, we see:

```
        1: other facilities, such as extra memory,
        2: or extra input and output channels. Our
        3: dealer may be able to give more details.
*_
```

Note that the lines left in the memory have been renumbered. So we should keep close track of which part of the file we are working with.

If we store these lines later, they will be added to what we sent back to the disk, so nothing will get lost.

6.13: Appending - A

After creating some room in the memory by saving a section of our MEMO file, as we just did, we can call up an additional part from the disk with an 'a' command (for 'append'). Thus:

```
*a <RET>
```

EDLIN will then load as much as possible of the file, leaving some open space for its own operation.

We may also indicate how many lines we want appended. For instance:

```
*100a <RET>
```

If we see the 'End of input file' message, we know that the first hundred lines of the file are now in memory, apart from the beginning, which has been written away to the disk with the 'w' command.

In case no message appears, we must try and append less text, or write away a larger portion, to make enough space available for the last part.

6.14: Copying - C or M

So far we have not mentioned any command which makes EDLIN duplicate part of our file. There are two: the 'c', for 'copy', and 'm', for 'move'.

Using the MEMO file, for instance, we may give the following instruction:

```
*5,8,23,1c <RET>
```

The first two figures indicate the sequence of lines that we want to copy, in this case from line 5 to line 8 inclusive.

The next number indicates the place where we want to insert a copy of this block. In this case the place is line 23, causing the original 23rd line and the following ones to move up to a higher number, just as when the 'i' command was used (see page 100).

The last figure, in this case 1, indicates the number of copies of the block we want EDLIN to make.

We may leave any of the numbers out, but not the commas. Leaving out the first, the second, or the third number makes EDLIN take the active line as the beginning, the end, or the destination of the block. Leaving out the last number makes EDLIN copy the block just once.

The 'm' command, for 'move', works in the same way as 'c'. The difference is that with 'c' a block is copied, leaving the original where it was, and with 'm' the block is moved.

Also, an 'm' command is always executed just once, so we do not need the last comma and number. An example:

```
*,23,7m <RET>
```

This command will move a block of lines from the active line, up to and including line 23, to the position of line 7.

6.15: Reading in - T

We can use the command 't' (for 'transfer') to add the contents of another file to the one we are working on.

Suppose we have a file called NOTES on the disk in drive A, and we want to insert it into the file we are editing. The place of insertion should be the current line 16. In that case we must type the following command:

```
*16ta:notes <RET>
*_
```

The original NOTES file is still on the disk in drive A, but its contents are now also part of the file we are presently working on.

7: DEVICE DRIVERS

As mentioned on page 13, the actual computer consists of the CPU, the Central Processing Unit. The CPU comprises a board with electronics inside the computer cabinet, the so-called mother board, containing a large number of chips. The most important chip is the microprocessor.

All the other parts of the computer should be considered as extra instruments and facilities. Examples are the keyboard that operates as a hatch for our commands, the screen as a projector, the operating memory or RAM as a workshop and temporary storage, and the external memory, that is, the hard disk or the floppy disks on which the results of the operations and instructions can be stored. We have discussed most of these parts in some detail.

Besides these indispensable accessories, the computer is equipped with a number of other extras. We may think of a printer, a colour screen or a mouse. In computerese, these extra parts and facilities are known as devices. They have in common that they are either the source or the destination of the data the CPU is processing. In other words, they are the instruments for the input of data (keyboard, mouse, clock) or for the output (printer, screen) or for both (disk drives). That is why they are also known as I/O devices, in which I/O stands for Input/Output.

The connector between the various devices and the actual computer usually consists of a so-called adapter or controller.

An example is the colour/graphics adapter. Normally, it has the form of a rectangular card full of chips and electronics to be inserted in a specially designed slot in the mother board. This adapter effects the connection and the use of a colour screen. Another example is the hard disk controller, which is also a card to be inserted in a slot in the mother board, and which controls the connection and use of the hard disk.

In order to activate a device in the way desired, it has to be controlled. For this purpose, the accompanying adapter is provided with a kind of electronic connector or switch that operates as a remote control. This remote control works by means of a small program that is known as a device driver. For instance, if we want to instruct the output channel leading to the printer to get ready for the transport of text to be printed, the relevant device driver has to be activated. Such a device driver may therefore be considered a kind of miniature operating system.

Without realising it, we have already become acquainted with a number of device drivers when we discussed the input and output channels of the computer (see page 58). In the first place, the driver of the console which we can activate by means of the term CON:. Second, the driver of the parallel channel with the activating terms PRN: and LPT1:. And finally, the driver of the serial channel with the activating terms AUX: and COM1:. These device drivers constitute a part of the programming code in the ROM BIOS (see page 25) and are therefore inextricably connected to the computer.

Other types of device drivers have to be loaded individually if they are to perform their tasks. To do so, we have to give the relevant command during or after the loading procedure.

As a result, a small amount of extra programming codes will be stored, in addition to the system, in the computer's operating memory, where it establishes a seamless connection with the loaded part of the operating system.

7.1: The screen

The most conspicuous part of the computer is the screen. At first sight, there are just two types. In the first place, the monochrome screen that displays characters in one colour only. Secondly, the colour screen, which can display characters in several different colours.

Apart from the fact that a monochrome screen can only display one colour, there is another disadvantage compared to the colour screen: it can only display text. In other words, the characters from the ASCII table (see page 68) and the extended character set (see page 75).

The only change we can make is in the maximum number of characters on the screen. We can choose between a screen of 40 characters by 25 lines, or a screen of 80 characters by 25 lines. The latter is the default, that is, the normal configuration. If we want to switch between these two configurations we have to activate a device driver which is present in the MODE.COM file.

For instance, if we want to switch from 80 x 25 to 40 x 25, we type the following command, assuming that MODE.COM is present in the root directory of drive C:

C>mode 40 <RET>

Immediately after we have hit the RETURN key, the characters, if any, disappear from the screen, and the prompt appears in the top left-hand corner, but larger than before:

C>_

The characters following it appear on the screen in the same size.

If desired, we use the same MODE command to give the instruction to move the display a bit to the left (l) or to the right (r), possibly preceded by a test pattern (t). This may be of importance if the screen does not function properly or if we use a television set as screen.

For instance, if we want to move the display to the left, plus a test to see whether everything is in order, we type:

C>mode l,t <RET>

As a result, the following test pattern appears on the screen:

012345678901234567890123456789

Then we have to answer this question:

Do you see the rightmost 9?

If we type a 'y', the prompt appears on the screen again.

On the other hand, if we type an 'n', the display moves again, followed by the same question. And again we can type an 'n', and so on, until the display is entirely to our satisfaction.

To move the display to the right, we start the procedure as follows:

C>mode 40,r,t <RET>

The result:

01234567890123456789012345678

Do you see the leftmost 0?

The same applies: if we type a 'y' the prompt appears on the screen, if we type an 'n' we see the test pattern and the question.

We can cancel the 40 x 25 display by means of a RESET procedure (see page 18), or by the reverse command:

C>mode 80 <RET>

Immediately after hitting the RETURN key, the prompt appears in the top left-hand corner of the screen, but now in the normal size.

Apart from characters, the colour screen can also display graphics. This term includes all visual effects that cannot be regarded as text, such as arithmetical figures, drawings, diagrams and all characters that are not in the ASCII table and the extended character set. In that case, however, the computer has to be equipped with a colour/graphics adapter (see page 119).

The default, or the normal configuration of the screen, is the so-called text mode. This means that the same characters can be displayed as on a monochrome screen (see the previous page) but then colours are added.

If we want to maintain the text mode but leave out the colour, we give the following command:

C>mode bw80 <RET>

At the same time, we can replace the text mode by the graphics mode. To do so, we type the following:

C>mode bw40 <RET>

We regain the default configuration by typing:

C>mode co80 <RET>

Ever since the introduction of the IBM PC, all kinds of programs like Lotus 1-2-3 have been put on the market in which both text and graphic effects can be used. In order to enable users of a computer with a monochrome screen to use such software, numerous companies, of which Hercules Computer Technology is very well-known, sell special graphics adapters. They enable a normal, monochrome screen to display graphics.

If the computer is equipped with either a colour/graphics adapter or a graphics adapter like the one of Hercules, we have to give the following MODE command to enable the display of graphics:

C>mode co40 <RET>

7.2: The printer

When we choose a printer, we are confronted with a large number of types and makes, such as the dot matrix printer, the daisy-wheel printer, the plotter and the laser-beam printer. The computer, however, distinguishes only two types, and it depends on the output channel which type we are dealing with.

If we have chosen a printer that can only be connected to the parallel channel (see page 59), as is usually the case with a dot matrix printer, we only have to take care of the right printer cable and the right connection between the two machines. No further installation is needed.

We may also be dealing with a printer that only wishes to receive data via a so-called serial channel, for instance a daisy-wheel printer. If we indeed have that type of printer, we have to make clear to the computer that the output of data should be run via a serial channel instead of a parallel channel. Such a serial channel is indicated by the terms AUX: or COM1: or COM2:, as we saw on page 59.

If we want to install a serial channel, we first have to use the MODE utility we became acquainted with on the previous pages, before we turn on the printer, since the MODE utility also contains a device driver for these types of output channels.

Assuming that the MODE.COM file is in the root directory on the hard disk (drive C), we begin by typing the following command:

```
C>mode com1:96,n,8,1,p <RET>
```

As a result, we should receive a confirmation like the following on the screen:

```
Resident portion of MODE installed
COM1:9600,N,8,1,P

C>_
```

Next, we have to give a second MODE command:

```
C>mode lpt1:=com1 <RET>
```

If the COM1 channel is already occupied, for instance by a modem, we type:

```
C>mode lpt1:=com2 <RET>
```

Note that neither COM1 nor COM2 is followed by a colon in these formulas.

The following confirmation should then be the result:

```
LPT1: redirected to COM1
```

We have to install a serial channel each time we load the system. It may therefore be obvious that we store the two MODE commands in a AUTOEXEC.BAT file. On page 148 we will explain how this is done.

If the printer is connected to one of the parallel channels, for instance PRN: or LPT1:, we do not need the MODE utility, as we saw on the previous page. If we want, we can use it to change the maximum number of characters on each line that is sent to the printer.

In the standard setting, the system sends 80 characters per line and 6 lines per inch to the printer. If we use paper with a length of 11 inches, then each sheet contains 66 lines. These values can again be changed into 132 characters per line and/or 8 lines per inch.

Let's assume that the printer is connected to the LPT1: channel and that we want to change both values. In that case we have to formulate the MODE command as follows:

```
C>mode lpt1: 132,8 <RET>
```

The procedure is confirmed by the following message on the screen:

```
Resident portion of MODE loaded

LPT1: not redirected

LPT1: set for 132

Printer lines per inch set
```

We have to bear in mind, however, that this setting is only useful if we have a printer that can print 132 characters per line.

Should we want to change the number of characters per line, but leave the number of lines per inch the same, we type:

```
C>mode lpt1: 132 <RET>
```

The system's confirmation is:

```
LPT1: not redirected

LPT1: set for 132
```

If it only concerns the number of lines per inch, the formula will be:

```
C>mode lpt1:,8 <RET>
```

The system will then confirm the procedure like this:

```
LPT1: not redirected

Printer lines per inch set

C>_
```

Each time we turn on the computer we have to reinstall the printer setting. Therefore, it is preferable to store such a MODE command in a batch program (see page 145 and subsequent pages).

7.3: Remote control - CTTY

A rather mysterious device driver is called CTTY (for Change Teletype). It is not included in a utility file but in the hidden files of the system (see page 48). This device driver enables us to have our computer controlled by another console, provided that we have taken the necessary measures in advance. Distance does not play any part. The control machine may be in the same room or on the other side of the world.

Let's assume that the computer is connected to a printer with its own keyboard by means of a printer cable. First, we have to open a communication channel between the two machines. Since it concerns two-way communication, we have to use a serial channel, for instance the COM2: channel.

To establish the technical details of the communication we type the following formula, assuming that the MODE.COM file is in the root directory of drive C:

```
C>mode com1:96,n,8,1,p <RET>
```

The various values and letters we include in this formula depend on the target and the connection. In this case, since the machines are connected by a direct cable, we can use the highest baud rate (9600, abbreviated to 96). This results in the confirmation:

```
COM1:9600,N,8,1,p
```

Next, we give a second MODE command, as we did with the installation of a serial printer channel (see the previous pages), namely:

```
C>mode lpt1:=com2 <RET>
```

Again, this is confirmed:

```
LPT1: redirected to COM2
```

As soon as the connections are in order and the channels open, we can transfer control to the printer by means of the following command:

```
C>ctty com2 <RET>

_
```

As a result, the prompt is printed on paper and we can use the keyboard of the printer to type any command.

If we want to return the control to the computer itself, i.e. to the console or CON, we type:

```
ctty con <RET>
```

As a result, the prompt appears on the screen again.

Suppose that the machine we want to transfer control to is another computer. The procedure to be followed is more complicated, for the communication has to be carried out via the telephone with a so-called auto-answer modem on either side that can answer the phone automatically. Thus, one modem has to be connected to the host computer, and another to the remote machine.

If the connection between these machines is in order, we have to write a batch program. How to do this will be explained on page 145. We could name it REMOTE.BAT, for instance, containing the following:

```
mode com2:12,n,8,1,p
ctty com2
```

The exact letters and numbers in this formula depend on the type of modem and the kind of connection. Modems sometimes have a baud rate of 300, but usually 1200 (12) or 2400 (24).

The next step is to turn on the auto-answer modem in a way that it can answer the telephone on the first ring.

Finally, we run the batch program. If MODE.COM is on the disk in drive C, the following will appear on the screen:

```
C>remote

C>mode com2:12,n,8,1,p
COM2:1200,N,8,1,p

C>ctty com2

_
```

As soon as the CTTY command appears on the screen, the keyboard is out of operation, although the cursor continues flickering. The function of the keyboard is taken over by what we may call the 'receive data' line of the COM2 channel, and that of the screen by the 'send data' line.

There are two ways to restore the keyboard's control. First, to switch off the computer and immediately turn it on again. Second, to send a command via the COM2 channel we have just opened, namely:

```
ctty con <RET>

C>_
```

Let's take a closer look at the remote control computer. In order to enable it to take control, we have to open a compatible communications channel, establish the same technical details and then install the modem.

Then we call the first computer, i.e. the host. The modem picks up the line and makes the actual connection. If all goes well, the prompt will appear on the screen of the remote control computer. From that moment, we can type any command on the keyboard.

There are two ways to put an end to the remote control. If we simply hang up the telephone, the same thing happens to the host's modem on the other side. In this case we can start the procedure again if we want to.

On the other hand, if we give a CTTY CON command, the program is ended, resulting in the fact that the host cannot be reached via the modem anymore.

7.4: The CONFIG.SYS file

During the loading procedure the operating system searches the directory of the system disk in drive A or the root directory of the hard disk for the presence of a file named CONFIG.SYS (a combination of the terms System and Configuration). It serves to store the so-called installable device drivers that are connected to the system during the loading procedure.

For instance, suppose we want to provide the computer with a clock and calendar card so that we do not have to enter the date and time ourselves. A utility to connect the device driver to the operating system is usually supplied together with such a card.

Assuming that the name of the extra utility is CLOCK.SYS and is present on the system disk or on the hard disk, we create a CONFIG.SYS file by means of the EDLIN method (see the previous chapter) or the COPY.CON: method (see page 50). We include the following contents in the file:

```
device=clock.sys
```

We include both the CONFIG.SYS file and the CLOCK.SYS file in the system disk or, if the computer has a hard disk, in the root directory of drive C.

If we now reload the system, the code will be added to the system in the CLOCK.SYS file driver. With some drivers (though not with all) we receive a confirmation like:

```
CLOCK.SYS installed
```

Anyhow, the clock and calendar card from this example are installed without a problem. From now on, we always see the right date and time when we are loading the system, so that we do not have to enter them ourselves.

We may also include some changes to the standard configuration of the system itself in a CONFIG.SYS file.

An example is the number of files allowed to be concurrently open at the same time. In the default setting, the system only permits the use of eight permanent or temporary files at the same time. Five of them are already reserved for various functions of the system itself, which leaves only three more files for other tasks. This may be sufficient for relatively simple programs, but this is not the case for most professional software. Software like dBASE III, for instance, needs an absolute minimum of 20 concurrently open files.

Therefore, we have to instruct the system to accept a larger amount, with a maximum of 20 for versions of the system prior to 3.30, and a maximum of 255 for versions 3.30, 3.31 and later. To do so, we have to add a FILES= instruction to the CONFIG.SYS file, for instance FILES=20. Assuming that we want to maintain the CLOCK.SYS instructions we have just included, we put the following lines in the file:

```
device=clock.sys
files=20
```

We may also add a specification of the number of buffers to a CONFIG.SYS file. In the default setting, the operating system reserves two buffers in the computer's memory, each of 528 bytes. Data and program fragments we might need again will be stored in these buffers by means of an ingenious procedure. If we work with a program in which data is often transferred from and to a floppy disk or hard disk, an increased number of buffers will speed up work considerably.

The number of buffers allowed amounts to a minimum of 2 and a maximum of 99. A series of experiments, however, has shown that 15 buffers usually is the right amount, and that the system hardly performs any better with a greater amount. To give the CONFIG.SYS file the instruction to reserve 15 buffers, we type the following:

```
buffers=15
```

We can add the instruction to an already existing CONFIG.SYS file as well. For instance:

```
device=clock.sys
files=20
buffers=15
```

The order of the various instructions is not important.

Another instruction we can safely add to the CONFIG.SYS file is the expansion of the number of reserved letters for disk drives. On page 91 we have already seen that the default, or normal, configuration has a maximum of five drives, with the letters A to E.

Should we want to increase this maximum to eight, that is, up to and including H, we must add the the following contents to a CONFIG.SYS file:

```
lastdrive=h
```

In this case, too, we may add the instruction to the CONFIG.SYS file:

```
device=clock.sys
files=20
buffers=15
lastdrive=h
```

And finally, we can control the ^C and ^<BREAK> instructions in a CONFIG.SYS file. In the default, or normal setting, the computer every now and then (i.e. after a short period of time) checks if we have given such an instruction. However, in some cases it may be too late; it would have been better if the computer had responded to the instruction immediately.

We can instruct the computer to respond to a ^C or ^<BREAK> instruction immediately by adding the following command to the CONFIG.SYS file:

```
break=on
```

To cancel this instruction at a later stage, we need not alter or erase the CONFIG.SYS file and reload the system. The following command will suffice:

```
C>break off <RET>
```

7.5: Installing a RAM disk

One of the most important additions to the 3-versions of the system is the special RAMDISK or VDISK (for Virtual Disk) device driver. It enables to install a RAM disk (or electronic disk, or virtual disk). Such a virtual disk drive consists of a part of the computer's operating memory, or RAM, that emulates a normal disk drive, only much faster.

In order to install a RAM disk or virtual disk, we create a CONFIG.SYS file by means of EDLIN (see the previous chapter) or the COPY CON: procedure (see page 50) using the following instruction:

```
device=vdisk.sys
```

We may add the instruction to the contents of an already existing CONFIG.SYS file as well. For instance:

```
device=clock.sys
files=20
buffers=15
lastdrive=h
break=on
device=vdisk.sys
```

Anyhow, if we reload the system, assuming that the VDISK.SYS file is on the system disk in drive A or in the root directory of drive C, the instruction results in the installation of a RAM disk with a storage capacity of 64 Kb, as the confirmation shows:

```
VDISK Version 3.3 virtual disk D:
   Buffer size:         64 KB
   Sector size:        128
   Directory entries:   64
```

Note that the virtual disk in this example has been assigned the letter D.

If the computer is not equipped with a hard disk (which is automatically named drive C) but with two floppy disk drives, the indication of the virtual disk will consist of the first letter that is available, hence the C.

As soon as the loading procedure is finished, we can use the RAM disk as a normal disk drive. A formatting procedure is not needed. Assuming that the virtual disk drive has been assigned the letter D, a CHKDSK command will result in the following information:

```
Volume VDISK V3.3 created Dec 6, 1984 12:00p

    62592 bytes total disk space
        0 bytes in 1 hidden files
    62592 bytes available on disk

   655360 bytes total memory
   551184 bytes free
```

If we want to install a RAM disk with a different size, we simply add the desired size (in kilobytes) to the instruction. So, for a RAM disk with a storage capacity of 400 Kb we type:

```
device=vdisk.sys 400
```

After loading the system, we see the following confirmation on the screen:

```
VDISK Version 3.3 virtual disk D:
   Buffer size:        400 KB
   Sector size:        128
   Directory entries:   64
```

It is possible to install two or more RAM disks. All we have to do is put the necessary number of DEVICE=VDISK.SYS commands in the CONFIG.SYS file.

We may also alter the size of the sectors of any of these virtual disks. The default value is 128 bytes; the alternatives are 256 and 512.

Moreover, we may alter the capacity of the directory of the virtual disks (normally 64 files) to any number between 2 and 512.

Now suppose we want to install two virtual disks: one with a storage capacity of 400 Kb, a sector size of 512 bytes and a maximum of 128 directory entries, and the other with a size of 220 Kb and for the rest the default values. To accomplish this, we have to include the following commands in the CONFIG.SYS file:

```
device=vdisk.sys 400 512 128
device=vdisk.sys 220
```

After reloading the system, we will see messages like the following on the screen, assuming that the size of the operating memory is 640 Kb:

```
VDISK Version 3.3 virtual disk D:
   Buffer size:        400 KB
   Sector size:        512
   Directory entries:  128

VDISK Version 3.3 virtual disk E:
   Buffer size adjusted
   Sector size adjusted
   Directory entries adjusted
   Buffer size:        143 KB
   Sector size:        128
   Directory entries:   64
```

Note that the 'buffer size' of virtual disk drive E has been 'adjusted' to 143 Kb instead of the 220 Kb wanted. That is because the VDISK program always leaves a minimum of 64 Kb of operating memory.

If the computer is equipped with an extended or expanded memory, we cannot use the part of the capacity over 640 Kb for our normal operations. However, we can install a virtual disk in it. To do so, we have to use the parameter /e (for extended memory). A correct formula would be, for instance:

```
device=vdisk.sys 400 128 256/e
```

7.6: Entering and changing date and time (2)

The version of PC-DOS or MS-DOS what we are working with is in most cases designed for use in Europe. If it is, then we can adjust the lay-out of the keyboard, and in particular the date and time notations, to non-American conventions and customs. For this purpose, a device driver is available that we can store in a CONFIG.SYS file by means of an instruction, just like we did with the examples on the previous pages.

To effect such adjustments, we can follow two routes. The first consists of creating a CONFIG.SYS file including a COUNTRY instruction, followed by a code number indicating the convention we want. We may choose a number from the following list:

country:	code:	country:	code:
United States	001	Great Britain	044
		Denmark	045
The Netherlands ..	031	Sweden	046
Belgium	032	Norway	047
France	033	Germany	049
Spain	034	Australia	061
Italy	039	Finland	358
Switzerland	041	Israel	972

Suppose we want to adopt the German system. In that case we create a CONFIG. SYS file including, in any case, the following command:

```
country=049
```

If we now reload the system, the device driver is taken up in the computer's operating memory and connected to the system.

The result is that the division sign in the date display is a full stop instead of a hyphen, and that, on entering the date, the system will stick to the order used in Germany (day-month-year):

```
Current date is Tue  1.01.1980
Enter new date (dd-mm-yy): _
```

In the display and registration of the time, the original division sign, the colon, is replaced by the full stop, and the full stop by a comma. Thus:

```
Current time is 0.00.37,18
Enter new time: _
```

A DIR instruction reveals that the date and time notations in the directory are organised accordingly. In addition, the time registration follows the continental 24-hour-system instead of the American 12-hour-system.

By simply replacing COUNTRY=049 by COUNTRY=044 in the CONFIG.SYS file, we can adopt the British convention. This will result in the following:

```
Current date is Tue  1-01-1980
Enter new date (dd-mm-yy): _

Current time is 0:00:41.26
Enter new time: _
```

In a similar way, we can adopt the Belgian, French, Spanish or Italian notation, each resulting in the following:

```
Current date is Tue  1/01/1980
Enter new date (dd-mm-yy): _

Current time is 0:00:39,22
Enter new time: _
```

If we take the Swiss option, we will see:

```
Current date is Tue  1.01.1980
Enter new date (dd-mm-yy): _

Current time is 0.00.40.24
Enter new time: _
```

The results of the Dutch and Finnish notation are:

```
Current date is Tue  1-01-1980
Enter new date (dd-mm-yy): _

Current time is 0:00:38,20
Enter new time: _
```

The Danish:

```
Current date is Tue  1/01/1980
Enter new date (dd-mm-yy): _

Current time is 0.00.42,28
Enter new time: _
```

The Swedish:

```
Current date is Tue  1980-01-01
Enter new date (yy-mm-dd): _

Current time is 0.00.43,30
Enter new time: _
```

And the Israeli notation:

```
Current date is Tue  1/01/1980
Enter new date (dd-mm-yy): _

Current time is 0:00:44.32
Enter new time: _
```

7.7: The keyboard (3)

To show programmers how a device driver can be connected to the system by means of a CONFIG.SYS file, IBM and Microsoft supplied PC-DOS and MS-DOS with an example named ANSI.SYS.

This device driver is an alternative for the normal console driver (see page 119) taken up in the ROM BIOS, and contains codes corresponding with the codes of the American National Standards Institute ANSI - hence the name. As soon as we adopt ANSI.SYS, the normal console driver will automatically be out of operation. The result is that we can install both devices in the way we please.

In order to install the device driver, we follow the same procedure as we did before. We create a CONFIG.SYS file and include the following command:

```
device=ansi.sys
```

If the system disk already contains a CONFIG.SYS file, we simply add the above instruction by means of EDLIN or the COPY CON: procedure:

```
device=clock.sys
files=20
buffers=15
lastdrive=h
break=on
device=vdisk.sys 400
country=031
device=ansi.sys
```

As mentioned before, the order of the commands is of no importance.

If we load the system again, ANSI.SYS, assuming that it is on the system disk in drive A or in the root directory of the hard disk, takes over control of the keyboard and the screen. Then we are, for instance, capable of giving function keys a different meaning.

To do so, we have to follow the same procedure as when we operate the printer by means of BASIC commands, as we will see on page 155 and further: a combination of passing on certain codes that indicate their meanings, an ESC symbol for instance.

At first sight, we seem to be confronted with the problem that we cannot pass on an ESC symbol just like that, because pressing the ESC key results in a command being ignored instead of carried out. Now the PROMPT command, which we will discuss on page 152 in greater detail, turns out to be very useful, since we can pass on an ESC code by means of the '$' key and 'e' key. The instruction to give a key a different meaning is the following:

```
ESC [ xx ; "yyy" p
```

First, we have to pass on the ESC symbol, then the [sign, then the decimal ASCII value of the character chosen instead of xx, next a semicolon and, between inverted commas, the character or string in question instead of yyy, and finally a 'p' (for permanent).

Let's, by way of experiment, convert the small 'a' into a small 'b'. The decimal ASCII value of a is 97, as the table on page 68 shows. Thus:

```
C>prompt $e[97;"b"p <RET>
```

The screen stays empty. No wonder - we have just replaced the actual prompt by the formula we have just typed. We first have to set this straight:

```
prompt <RET>
```

If we now strike the 'a' key, a 'b' should appear on the screen.
   We can restore the original setting, in other words, get an 'a' after striking the 'a' key, in two ways. First, by starting a RESET procedure, and second, by using the following command, which restores the original situation:

```
C>prompt $e[97;97p <RET>
```

Note that we could not just have typed the 'a' in this formula - we have just replaced it by a 'b'. So we have to describe the 'a' by means of the decimal ASCII value. We then type:

```
prompt <RET>
```

Now everything is back to normal: the prompt reappears and an 'a' is an 'a'.

We can create extra possibilities by giving a combination of keys a specific meaning, for instance the CTRL key or the ALT key with the function keys, or the ALT key with a number of ordinary letter keys.
   To do so, we have to use the code list included in ANSI.SYS:

key(s):	code:	key(s):	code:	key(s):	code:	key(s):	code:
HOME	71	SHIFT+F1	84	CTRL+F1	94	ALT+F1	104
PG UP	73	SHIFT+F2	85	CTRL+F2	95	ALT+F2	105
END	79	SHIFT+F3	86	CTRL+F3	96	ALT+F3	106
PG DN	81	SHIFT+F4	87	CTRL+F4	97	ALT+F4	107
CTRL+HOME	119	SHIFT+F5	88	CTRL+F5	98	ALT+F5	108
		SHIFT+F6	89	CTRL+F6	99	ALT+F6	109
F7	65	SHIFT+F7	90	CTRL+F7	100	ALT+F7	110
F8	66	SHIFT+F8	91	CTRL+F8	101	ALT+F8	111
F9	67	SHIFT+F9	92	CTRL+F9	102	ALT+F9	112
F0	68	SHIFT+F0	93	CTRL+F0	103	ALT+F0	113
ALT+1	120	ALT+Q	16	ALT+A	30	ALT+Z	44
ALT+2	121	ALT+W	17	ALT+S	31	ALT+X	45
ALT+3	122	ALT+E	18	ALT+D	32	ALT+C	46
ALT+4	123	ALT+R	19	ALT+F	33	ALT+V	47
ALT+5	124	ALT+T	20	ALT+G	34	ALT+B	48
ALT+6	125	ALT+Y	21	ALT+H	35	ALT+N	49
ALT+7	126	ALT+U	22	ALT+J	36	ALT+M	50
ALT+8	127	ALT+I	23	ALT+K	37		
ALT+9	128	ALT+O	24	ALT+L	38	INS	82
ALT+0	129	ALT+P	25	DEL	83		

It is of course not practical to play around with keys that we have to use very frequently. It is more logical to give the function keys which have no specific task in the system (F7, F8, F9, F10) a meaning, or to change the significance of the keys we hardly ever use.

If we want to use one of the codes from the survey, it always has to be preceded by a zero and a semi-colon in the command. For instance, if we want to give the F8 key (code number 66) the meaning 'chkdsk b:', we type the following formula:

```
C>prompt $e[0;66;"chkdsk b:"p <RET>

prompt <RET>

C>_
```

If we then hit the F8 key, we see on the screen:

```
C>chkdsk b:
```

If we also add a semi-colon and the decimal ASCII code for the carriage return function (13) to the formula, we need not press the RETURN key after the command has been displayed. This code should, however, precede the 'p':

```
C>prompt $e[0;66;"chkdsk b:";13p <RET>

prompt <RET>

C>_
```

When we strike the F8 key, not only the text appears on the screen - the command is immediately carried out as well.

If we want to cancel this instruction again, we type:

```
C>prompt $e[0;66;0;66p <RET>

prompt <RET>

C>_
```

We may assign as many characters to each key as we want, as long as the total does not take more than 190 bytes, including spaces. If we exceed this number, the space they take up in RAM would spill over into the space needed for other things. In that case we will see the following error message:

```
Out of environment space
```

To have the keys with an altered meaning permanently at our disposal, it is advisable to store the formulas for these commands in a AUTOEXEC.BAT file or in a special batch program. We will return to this in the next chapter.

When the PS/2 version was introduced, all IBM computers, closely followed by all compatibles, were equipped with two extra function keys: F11 and F12. Not all software on the market takes these keys into consideration.

7.8: The screen dump

As we have already seen on page 16, the keyboard provides a special key marked PRT SC (for Print Screen). Pressing this key together with one of the SHIFT keys results in the printer producing a copy of the characters at that moment on the screen, the so-called screen dump - provided, of course, that the printer is connected to the computer and turned on.

As long as the screen displays only real ASCII signs, this will not cause any problem. But if it also contains extended characters (see page 75), such as lines, rasters or technical symbols, the printer will disappoint us. The special characters will usually be replaced by ordinary numbers and letters, ruining a beautiful framework, or a scientific formula.

In some cases a device driver which may be included in the files of the operating system under the name GRAPHICS.COM can be a solution. We activate it as follows, assuming that the file is to be found in the root directory:

```
C>graphics <RET>
```

There is no confirmation whatsoever, but something extra has been stored in the computer's memory, i.e. a table with the extended characters meant for the output of data through one of the printer channels.

The instruction only has effect if the printer is able actually to print such characters after having received the command to do so. This only applies to printers that accurately abide by the specifications of IBM, such as the IBM Graphics Printer, and the IBM Proprinter.

With the introduction of the 3-versions of PC-DOS, the device driver recognises a number of other types of IBM printers. In that case we have to extend the GRAPHICS command with an addition, depending on the type of printer used, namely:

COLOR1 - IBM Personal Computer Color Printer with black ribbon;
COLOR4 - same with RGB (red, green, blue, black) ribbon;
COLOR8 - same with CMY (cyan, magenta, yellow, black) ribbon;
COMPACT - IBM Personal Computer Compact Printer.

For instance, if we want the IBM Color Printer with a black ribbon to print a screen displaying extended characters, we type the following formula:

```
C>graphics color1 <RET>
```

With printers that just cannot print these extended characters, such as most daisy-wheel and laser-beam printers, the result remains the same as before.

We may add the parameter /r (for reverse) to the command. For instance:

```
C>graphics color1/r <RET>
```

In that case, everything that is dark or black on the screen will be printed in black and light will be printed in white. Any colours, solid surfaces or rasters are printed in a corresponding shade of grey.

7.9: Concurrent printing - PRINT

With the PRINT command we can instruct the computer to send a number of files (up to ten) to the printer, while remaining able to carry out other commands.

For instance, suppose the disk in drive A contains three text files (TEXT.1, TEXT.2 and TEXT.3) and that we want to have them all printed. Assuming that the PRINT.COM file is in the root directory of drive C, we type the following command:

```
C>print a:text.1 a:text.2 a:text.3 <RET>
```

Since wildcard symbols (see page 31) are allowed in a PRINT command, we can abbreviate the command as follows:

```
C>print a:text.?/p <RET>
```

The system's first reaction is to ask which device we wish to use:

```
Name of list device [PRN]: _
```

The default setting is the PRN: channel (see page 59). If this is the one we want to use, we only have to strike the RETURN key.

If we have already redirected output to a serial channel using the MODE command (see page 122), this setting will remain intact.

If we want to use another channel, however, we should type it in.

We may receive the following error message:

```
List output is not assigned to a device
```

Apparently, we have forgotten to turn on the printer, or we have made a mistake in typing the channel required. In any case, the procedure is halted.

Apart from the above error message, we can expect another one, namely:

```
Errors on list device indicate that it may be
off-line. Please check
```

Now we have to opportunity to check the printer's ON LINE-light. If it is off, we have to press the ON LINE-button.

As soon as everything appears to be in order, we see a confirming message such as:

```
Resident part of PRINT installed

        A:TEXT.1 is currently being printed
        A:TEXT.2 is in queue
        A:TEXT.3 is in queue
```

The printer will start operating immediately, while the prompt on the screen indicates that the system can be instructed to carry out another command. In other words, we have the opportunity to do something else in the mean time.

If the PRINT command is not entirely completed, we may add one or more files to the queue. For instance:

```
C>print a:text.4/p a:text.5/p <RET>
```

Note that we added the parameter /p (for print) to each filename.

As a result, the same list shown on the previous page returns to the screen, including the file(s) we have just added with the last command.

If we want to remove one or more files from the list, we give a PRINT command followed by the paremeter /c (for cancel) after each file we want removed. For instance:

```
C>print a:text.2/c a:text.3/c <RET>
```

Once again we see the list, this time without the files we have just cancelled.

If we cancel a file while it is being printed, the following message will appear on the paper:

```
File cancelled by operator
```

If we want to see how the PRINT procedure is progressing, we give the basic command, that is, without any additions. Thus:

```
C>print <RET>
```

The system's answer may be:

```
A:TEXT.1 is currently being printed
A:TEXT.4 is in queue
A:TEXT.5 is in queue
```

If a file cannot be found during a PRINT procedure, it will be removed from the list.

If we want to end the printing procedure, we have to issue a PRINT command with the parameter /t (for terminate):

```
C>print/t <RET>
```

As a result, the following message is printed:

```
All files cancelled by operator
```

On the screen we see:

```
Print queue is empty
```

8: STREAMLINING THE SYSTEM

Since the main task of a computer is to speed up time-consuming operations and to computerise ever-recurrent procedures, we may expect the operating system to be at our service. And indeed, it is. It provides various gadgets and devices to carry out all kinds of daily routine jobs in a fixed order without a problem. This chapter deals with some of these routine actions.

First, we will become acquainted with the computer term I/O Redirection. In other words, redirecting input and output. In fact, we have come across this before. On page 58 we saw various methods we can use to transfer files by means of COPY commands. On page 122 we saw how to install a serial channel by means of MODE commands, and how data is rerouted in this procedure.

The difference with these procedures is, however, that we now will see how we can procure such a redirection by just including a single character in a command, and how easily these commands can be carried out.

In addition, we will learn the meaning of a pipeline and how to create one - also by means of a single character.

Subsequently, we will become acquainted with the term 'filter'. This is an instruction for us to make a selection from an amount of data or to put such data in alphabetical or numeral order.

However, this is not entirely new to us either. On page 31 we learned how to work with wildcard symbols; actually, this was our first contact with a filter. Furthermore, we became acquainted with the MORE command on page 54. Although this command does not seem to be a filter, it does work as such. On the other hand, the filter function of the FIND procedure we saw on page 73 has been clear from the start.

This chapter will also deal with another filter command, namely SORT. As the name suggests, we can use it to put the (system-readable) contents of a text file in alphabetical or numeral order.

This whole structure of redirecting input and output, constructing pipelines and using filters will only fully develop if we combine the various elements. This chapter will provide us with an example of this: cataloguing a large numer of floppy disks.

Once we have become more familiar with the system and we have begun to develop a certain routine in our operations, we will be able to find numerous other possibilities to use this facility.

In this chapter we will extensively discuss the so-called batch program in general and the AUTOEXEC.BAT program in particular. This type of program enables us to have a number of ever-recurrent instructions carried out automatically. A nice touch is that we can compose, formulate and activate such a program relatively easily ourselves.

In fact, we take our first unsteady steps in the field of a programmer, for although the title of this chapter is rather innocent, it deals with the computerisation of our work - which just means programming.

To conclude the chapter, we return to the prompt command we already saw on page 133. It turns out to be able to fulfil an independent function which, as the name suggests, has a certain link with the system prompt.

8.1: Redirecting input and output

Redirecting input and output is quite normal in operating systems of the big ('mainframe') computers. In the system of a PC, however, it is not so normal, but the way it works is fairly simple.

To redirect input and output means that we have to indicate from where the system has to obtain certain information it has to process and where it has to send it to. We already saw an example of this on page 54 when we discussed the MORE command, namely:

C>more <a:notes <RET>

In fact, the instruction in this formula is: look for the NOTES file on the disk in drive A, consider it the source and reproduce the contents in sections of 23 lines on the screen. The < sign plays an important role in this formula: it marks the direction in which the data has to be transferred.

A second example is:

C>type a:notes >prn: <RET>

This command means: look for the contents of the NOTES file on the disk in drive A to be reproduced but do not send the result to the screen, as is usual in a TYPE procedure, but send it to the printer. So the > sign marks the direction of the transfer and the term PRN: marks the destination.

If the printer is properly connected to the computer and turned on, the contents of the NOTES file will now be printed on paper.

The available redirecting symbols and their meanings are:

> - to the following destination;
>> - addition to the following file;
< - from the following source.

Let's take another experiment. Normally, a CHKDSK command provides us with a survey of the storage capacity on a floppy disk or hard disk. In this example, however, we add a > sign to the command, followed by a file name, for instance CHKDSK.TXT and the letter of a drive, such as an A. Thus:

C>chkdsk >a:chkdsk.txt <RET>

There is no confirmation that the instruction has been carried out. Only the prompt appears on the screen again. If we call up the directory of the disk in drive A, however, a file named CHKDSK.TXT turns out to be included. Using a TYPE instruction, we can ascertain that the contents consist of the CHKDSK information that would otherwise have been displayed on the screen.

In a similar way we can store the directory of a disk in drive A in a file named DIR.TXT, for instance. Thus:

C>dir a: >dir.txt <RET>

In this case, too, there is no confirmation, and again, a TYPE command shows that the data has indeed been included in the file stated:

```
C>type dir.txt <RET>

 Volume in drive A has no label
 Directory of  A:\

NOTES                131  17-12-88  10:17
CHKDSK   TXT         307  29-12-88  16:34
        2 File(s)    360448 bytes free
```

Next, we replace the disk in drive A by another and repeat the command, but this time we add the >> sign instead of the > sign to the formula. Thus:

```
C>dir a: >>dir.txt <RET>
```

If we again check the contents of the DIR.TXT file, the data of the second disk turns out to have just been added to the first one:

```
C>type dir.txt <RET>

 Volume in drive A has no label
 Directory of  A:\

NOTES                131  17-12-88  10:17
CHKDSK   TXT         307  29-12-88  16:34
        2 File(s)    360448 bytes free

 Volume in drive A is 10605511100
 Directory of  A:\

DBASE    SER       18432  29-07-86  12:00
DBASE    EXE      133632   4-11-86  11:47
SIGNON   COM        2752  29-07-86  12:00
INITDB   BAT           2  29-07-86  12:00
CONFIG   DB           27  29-07-86  12:00
CONFIG   SYS          22  29-07-86  12:00
CONFI256 DB           97  29-07-86  12:00
CONFI256 SYS         128  29-07-86  12:00
DBASEINL OVL       27648  29-07-86  12:00
DBASE    MSG       12420   4-11-86  11:47
INSTALL  BAT        4608  29-07-86  12:00
UNINSTAL BAT        2816  29-07-86  12:00
README   TXT         845  29-07-86  12:00
YN       EXE         872  29-07-86  12:00
ID       EXE       16624  29-07-86  12:00
HELP     DBS       68608  29-07-86  12:00
       16 File(s)     25600 bytes free

C>_
```

In this way, we could add the directory of an almost unlimited number of disks to the DIR.TXT file. The next page will show the use of this.

8.2: Sorting - SORT

We may use the SORT utility if we want the contents of a text file sorted in a certain order.

However, not every file can be tackled with this tool. Its contents must contain so-called standard input or standard output. This means that the contents must have been created by a system utility, such as EDLIN. In other words, it has to consist of reproducible characters from the ASCII table and the extended character set.

Let's, by way of experiment, try to put the contents of the DIR.TXT file we have just created in alphabetical order. To do so, we type the following, assuming that the SORT.EXE file is in the root directory of drive C:

```
C>sort <dir.txt <RET>
```

Next, this should result in the following message on the screen:

```
        2 File(s)    360448 bytes free
       16 File(s)     25600 bytes free
 Directory of  A:\
 Directory of  A:\
 Volume in drive A is 10605511100
 Volume in drive A has no label
CHKDSK   TXT       307  12-29-88  16:34
CONFI256 DB         97  29-07-86  12:00
CONFI256 SYS       128  29-07-86  12:00
CONFIG   DB         27  29-07-86  12:00
CONFIG   SYS        22  29-07-86  12:00
DBASE    EXE    133632   4-11-86  11:47
DBASE    MSG     12420   4-11-86  11:47
DBASE    SER     18432  29-07-86  12:00
DBASEINL OVL     27648  29-07-86  12:00
HELP     DBS     68608  29-07-86  12:00
ID       EXE     16624  29-07-86  12:00
INITDB   BAT         2  29-07-86  12:00
INSTALL  BAT      4608  29-07-86  12:00
NOTES              131  12-17-87  10:17
README   TXT       845  29-07-86  12:00
SIGNON   COM      2752  29-07-86  12:00
UNINSTAL BAT      2816  29-07-86  12:00
YN       EXE       872  29-07-86  12:00
```

Note that the lines beginning with a space have been moved to the top.

The reason is that the space (the SP symbol) in the ASCII table (see page 68) has a lower value than the letter A.

Suppose we do not only want to take a look at the result of a SORT procedure, but also store it in a file named ALPHABET, for instance, we have to add a second redirecting symbol to the formula as follows:

```
C>sort <dir.txt >alphabet <RET>
```

In order to specify a SORT command in greater detail, we may add one or more parameters, for instance /r, for reverse. This results in the lines being arranged in a reverse alphabetical order. Thus:

```
C>sort <dir.txt/r <RET>
```

To store this data in a file named ALPHABET2, for instance, we type:

```
C>sort <dir.txt >alphabet2/r <RET>
```

A second parameter we can add is /+, followed by a specific number. As a result, the directory is not sorted on the first letter of the line, but on the letter in the column indicated by the number.

Suppose we want to sort the file names in byte size order, instead of alphabetical order. A quick count shows that the file size statements start on the thirteenth column. Therefore, we should type the following formula:

```
C>sort/+13 <dir.txt <RET>
```

```
INITDB   BAT         2  29-07-86  12:00
CONFIG   SYS        22  29-07-86  12:00
CONFIG   DB         27  29-07-86  12:00
CONFI256 DB         97  29-07-86  12:00
CONFI256 SYS       128  29-07-86  12:00
NOTES              131  17-12-88  10:17
CHKDSK   TXT       307  29-12-88  16:34
README   TXT       845  29-07-86  12:00
YN       EXE       872  29-07-86  12:00
SIGNON   COM      2752  29-07-86  12:00
UNINSTAL BAT      2816  29-07-86  12:00
INSTALL  BAT      4608  29-07-86  12:00
DBASE    MSG     12420   4-11-86  11:47
ID       EXE     16624  29-07-86  12:00
DBASE    SER     18432  29-07-86  12:00
DBASEINL OVL     27648  29-07-86  12:00
HELP     DBS     68608  29-07-86  12:00
DBASE    EXE    133632   4-11-86  11:47
 Directory of  A:\
 Directory of  A:\
       16 File(s)     25600 bytes free
        2 file(s)    360448 bytes free
 Volume in drive A has no label
 Volume in drive A is 10605511100
```

Again we can store this data in a file if we like. Consequently, we have become acquainted with the procedure to have the system sort the names of the files on a number of disks in the order of alphabet, size, date or time.

During a SORT procedure two temporary files are created, in which the actual sorting takes place. Afterwards they are automatically deleted. We have to make sure that there is enough space on our floppy disk or hard disk containing the SORT.EXE file to store these two temporary files.

8.3: The pipeline

On the previous pages we met a number of filter procedures and the redirecting symbols >, >> and <. If we add the | sign as well, we have all the instruments at our disposal to make constructions that are known as 'pipes'.

Let's once again face the problem of how to catalogue a number of disks. We have seen that we need various commands, namely:

```
C>dir a: >dir.txt <RET>

C>dir a: >>dir.txt <RET>

C>dir a: >>dir.txt <RET>
```

And so on, until we have catalogued all disks. Then we type for instance:

```
C>sort/+13 <dir.txt >files.txt <RET>
```

Finally, the sorted data will be stored in the file named FILES.TXT.

By means of the | sign, we can pass on the data directly so that we need not create the DIR.TXT file. In that case the command should be:

```
C>dir b:|sort >>files.txt <RET>
```

Note the function of the | sign in the command: it fences off the preceding command, announces the next one and passes on the collected data. This constitutes a pipeline.

Once the procedure is finished, we replace the disk in drive A by another and strike the F3 key (see page 22). As a result, the command reappears on the screen and is executed when we strike the RETURN key. We repeat this procedure until we have tackled all the disks we want to have catalogued.

Our second step is to filter the file. We remove the lines containing the word 'File(s)' from the FILES.TXT file and store the result in a definitive file named CONTENTS. To do so, we use the FIND command (see page 73) as follows:

```
C>find/v 'File(s)' files.txt >contents <RET>
```

Finally, we check the result by means of a TYPE or MORE command:

```
C>more <contents <RET>
```

Thanks to the possibilities offered by a pipeline procedure, we can combine the first and second step as follows:

```
C>dir a:|sort|find/v 'File(s)' >>contents <RET>
```

This command can also be repeated for each disk in drive A by means of the simple combination of the <F3> and <RETURN> keys.

8.4: The batch program

There is a chance that, in due course, we repeatedly have to formulate and pass on the same commands to carry out certain procedures. In order to avoid typing them again and again and to avoid the risk of making typing errors, we can make use of the fact that the system is capable of storing a number of commands and executing them automatically. To achieve this, we have to include these commands in a so-called batch program on the system disk. The system sees to it that the commands are being executed one by one in the required order.

For instance, let's assume we have connected a printer to a serial channel, such as the COM1: channel. In that case, we have to type two MODE commands very accurately each time we want to use the printer, as we have already seen on page 122. In this case it seems sensible to include these two commands in a batch program, which we may call PRINTER.BAT.

To create this file we can either use EDLIN (see page 95) or the COPY CON: procedure (see page 50). In any case, assuming that the MODE.COM file is in the root diretory of the hard disk, the contents should be:

```
mode com1:96,n,8,1,p
mode lpt1:=com1
```

Once the file has been stored we can put it into effect at any time. Assuming that it is in the root directory of the hard disk, we then type:

```
C>printer <RET>
```

We do not need to type the extension, just like when we invoke .COM and .EXE files. The text of the commands appears on the screen, directly followed by the results, as if we had typed and passed on the commands one by one:

```
C>mode com1:96,n,8,1,p

COM1:9600,N,8,1,P

C>mode lpt1:=com1

LPT1: redirected to COM1
```

As soon as all the commands in the batch program have been carried out, the prompt reappears on the screen, after which we can take up our work again.

If we want to break off the procedure for any reason, we have to resort to a ^C or ^<BREAK> instruction. On the screen we see:

```
Terminate batch job (Y/N)? _
```

If we type a 'y', the program will be terminated definitively, after which the prompt reappears on the screen. However, if we type an 'n', the program will continue where it left off and carry out the procedure.

Suppose we are used to correcting and supplementing our NOTES file regularly and that we want to view it and possibly print it once a day.

In that case it seems sensible that we create a batch program which we may name MEMO.BAT. The contents must be the following:

```
more <a:notes
copy a:notes prn:
```

We may precede it by the commands from our PRINTER.BAT file so that we only need one batch program. Thus:

```
mode com1:96,n,8,1,p
mode lpt1:=com1
more <a:notes
copy a:notes prn:
```

As soon as this batch program has been stored, we only have to type MEMO in order to have it operated. First we see how the MODE commands are being carried out, and next the MORE command appears on the screen:

```
C>more <a:notes <RET>
```

This is followed by the first 23 lines of the NOTES files, and so on.

Then the COPY command is displayed, after which the contents of the file are sent to the printer via the PRN: channel (see page 59). If the file has been printed, we see:

```
1 File(s) copied
```

8.5: Comments and interruptions

A drawback of the above example is that we receive an error message in those cases where we have forgotten to insert the disk in drive A or if it does not contain a file named NOTES, or if we cannot call up the MORE.COM file. Moreover, the computer can get confused if the printer has not been turned on or not properly connected to the computer.

In order to overcome such contingencies, the system offers us the possibility of adding a REM command (for remark) to the batch program, followed by a message of no more than 123 characters. We use it to alert ourselves or someone else using the computer. For instance:

```
rem - Is the disk with NOTES in drive A?
rem - Is the printer connected?
more <a:notes
copy a:notes prn:
```

If we start the revised version of the batch program, the two REM messages will be displayed first:

```
C>rem - Is the disk with NOTES in drive A?
```

```
C>rem - Is the printer connected?
```

For the rest, the program works exactly the same as it did before.

A disadvantage of this construction is that we have to follow the commands that are being carried out on the screen very carefully, so that we can interrupt the procedure by means of a ^C or ^<BREAK> instruction, should this be necessary.

However, we can overcome this disadvantage by using the PAUSE command. This command results in the procedure being halted, and it will only be continued after we strike a key. Let's adjust the program as follows:

```
rem - Is the disk with NOTES in drive A?
rem - Is the printer connected?
pause
more <a:notes
copy a:notes prn:
```

If we run the program in this form, the following message, after the REM messages have been displayed, will appear on the screen:

```
C>pause
Strike a key when ready . . . _
```

Now we have the opportunity to insert the right disk in drive A or to check whether the printer is connected. If we then strike a key, the program will continue where it left off. Should we nevertheless wish to terminate the program, we have to give a ^C or ^<BREAK> instruction (see page 71).

We have the possibility of combining the PAUSE and REM commands, for we may extend PAUSE by a text to a maximum of 125 characters. For instance:

```
pause - Is the disk with NOTES in drive A?
pause - Is the printer connected?
```

As the program is being executed we see:

```
C>pause - Is the disk with NOTES in drive A?
Strike a key when ready . . . _
```

Instead of REM, we can also use the ECHO command. The result is exactly the same, but ECHO has another function as well. If we open the series of commands with ECHO OFF, and close it with ECHO ON, we avoid the command formulas being displayed one-by-one on the screen as they are carried out.

Between these commands, we should not precede any comment messages by PAUSE or REM but by ECHO, which, unlike REM and PAUSE, circumvents the ECHO OFF command. Thus:

```
echo off
echo Is the disk with NOTES in drive A?
echo Is the printer connected?
pause
more <a:notes
copy a:notes prn:
echo on
```

8.6: The AUTOEXEC.BAT file

A special kind of batch program is a file named AUTOEXEC.BAT. In the loading process, the system not only looks for a CONFIG.SYS file (see page 126) on the floppy disk in drive A or the hard disk, but also for an AUTOEXEC.BAT file. And if present, the system automatically puts it into operation.

We can create an AUTOEXEC.BAT file in the same way as any other batch program. This will be clear when we rename our MEMO.BAT to AUTOEXEC.BAT. If we now start a RESET procedure, resulting in the system being reloaded, the file is automatically put into operation, assuming of course that it is in the root directory of the hard disk, or on the system disk in drive A.

An AUTOEXEC.BAT program suppresses the usual system requests to enter the date and time. If the computer is not equipped with a clock/calendar device (see page 126), it is essential to include a DATE and TIME command in an AUTOEXEC program. For instance:

```
echo off
date
time
cls
echo Is the disk with NOTES in drive A?
echo Is the printer connected?
pause
type a:notes >prn:
echo on
```

As the commands are being carried out we have the opportunity to enter the date and the time before the first ECHO message is displayed on the screen. After the system has been loaded, we first see:

```
Current date is Wednesday 12-30-88
Enter new date (mm-dd-yy):_
```

As we have seen on page 122, we need to give two MODE commands before we can work with a printer connected to a serial channel. These two commands can be included in the AUTOEXEC.BAT file. In our example, this may result in the following:

```
echo off
mode com1:96,n,8,1,p
mode lpt1:=com1
date
time
cls
echo Is the disk with NOTES in drive A?
echo Is the printer connected?
pause
type a:notes >prn:
echo on
```

8.7: The condition

By means of the command IF EXIST we require a file to be present on a disk in a certain drive or in a certain directory. If this condition is not complied with, the program will continue in a different way to be further specified.

For instance:

```
if exist a:notes \more <a:notes
pause - No NOTES have been found. What next?
```

Note that the IF EXIST command is followed by another command on the same line. The second command on the line is only carried out if the required condition is complied with; in this case - if the file in question is present on the disk. If not, the command on the line following it will be carried out, resulting in the following message appearing on the screen:

```
No NOTES have been found. What next?
Strike a key when ready . . . _
```

A negative IF EXIST command is also possible. For instance:

```
if not exist a:notes echo No NOTES
pause What next?
```

We have the possibility to instruct the system to break off the batch program as soon as a condition is not complied with; in this case - if the file in question turns out not to be present on the disk. For this purpose we add the command EXIT. Thus:

```
if not exist a:notes exit
```

If the NOTES file does not appear to be present on the disk as the program is being run, the system prompt will reappear on the screen immediately. From that moment, not a single command in the batch program will be carried out.

Assuming that we have added this last formula, the program has meanwhile been given the following form.

```
echo off
date
time
cls
echo Is the disk with NOTES in drive A?
echo Is the printer connected?
pause
if not exist a:notes exit
type a:notes >prn:
echo on
```

8.8: The jump

Another utility originating from the world of programming languages, in particular BASIC, is the batch command GOTO. It enables us, usually after an IF EXIST command, to have the system jump backwards or forwards in the sequence of commands.

We can determine the destination of the jump with a so-called label, i.e. a word, a code or whatever, of no more than eight characters, preceded by a colon. For instance:

```
echo off
date
time
cls
:change
echo Is the disk with NOTES in drive A?
echo Is the printer connected?
pause
if not exist a:notes goto change
type a:notes >prn:
echo on
```

If there is no file on the disk in drive A called NOTES, the IF NOT EXIST condition has indeed been complied with. As a result, the next command on the same line will be carried out; in this case - perform a jump to the label called CHANGE.

The next step is that the first command following the label will be displayed. So again we see:

```
Is the disk with NOTES in drive A?
Is the printer connected?
Strike a key when ready . . . _
```

If we fail to insert the right disk in drive A, we get into a vicious circle with label and commands. Our only way out is a ^C or ^<BREAK> instruction.

It may be that we just wish to get another opportunity to have the disk in drive A searched for a specific file, for instance because several disks contain a file with this name.

In that case we have to include the affirmative IF EXIST command in the program. Thus:

```
:change
echo Insert a disk with NOTES in drive A
pause
if not exist a:notes goto end
if exist a:notes type a:notes >prn:
goto change
:end
echo on
```

8.9: Variables

Suppose we want to conclude a batch program with the command to use EDLIN (see page 95) and open the NOTES file at the same time. Thus:

```
echo off
:change
echo Is the disk with NOTES in drive A?
pause
if not exist a:notes goto change
edlin a:notes
echo on
```

The disadvantage of the program now is that we only can open a file named NOTES. It would be far more practical if we could change the name of the program at will, without affecting it.

Well, this is possible. For we may replace the file name in the program by the symbol of a so-called variable. A more detailed explanation will be given on page 162. In this example a variable can be included in the form of the % sign, followed by a numeral from 1 to 9. Consequently, we may change the second last line of the program as follows:

```
edlin a:%1
```

Assuming that it is named MEMO.BAT and present in the root directory of drive C, we type the following in order to put the program into operation:

```
C>memo notes <RET>
```

or:

```
C>memo text <RET>
```

Whatever the name we type, the system will put it in place of the symbol of the variable as the program is being run.

Another disadvantage is that we are restricted to a specific drive or sub-directory. We would benefit from the use of the variable if we would be able to include the letter of the drive or the name of the sub-directory in it as well. And indeed, this is also possible. We simply add a second variable:

```
edlin %1 %2
```

As a result of this formula we can put the program into effect by typing:

```
C>memo a:notes <RET>
```

or for instance:

```
C>memo \text \notes <RET>
```

8.10: The prompt

A nice touch, but at first sight little more than that, is the capability offered by version 2 of the system to alter the appearance of the prompt. Assume we would like to be greeted by:

At your service:_

Getting this done is simplicity itself. We just type the following command:

C>prompt At your service: <RET>

As soon as we strike the RETURN key, the prompt appears in the desired form.

This prompt works the same as usual, the only difference being the form. For instance, if we want to switch to drive B we type, as usual:

At your service:b: <RET>

The answer shows a drawback of the new prompt since there is no way of knowing whether the command has been carried out:

At your service:_

If we give a DIR command, however, B has indeed become the active drive.
There is a solution, however. We may supplement the PROMPT command with one or more special characters that all have a specific significance. These characters are listed below:

d = the system date
n = the active drive
p = the active sub-directory
t = the system time
v = the system version

b = the symbol |
g = the symbol >
l = the symbol <
q = the symbol =

_ = command for a new line
h = command for backspace (the same as ^H)
e = command to pass on an ESC symbol

In order to avoid confusion with the normal characters, we have to precede all these special characters by a dollar sign ($) in the formula.

This is the solution to the problem we just met:

At your service:prompt ($p) At your service:

As a result, the active drive or sub-directory now opens the prompt line:

```
(C:\) At your service:_
```

To make the prompt something really special, we might formulate the command as follows:

```
prompt This is $v.$_The system time now is $t.$_
The active subdirectory is $p.$_At your service:
```

Note that as we type this command, the cursor moves to a new line automatically when it reaches the right edge of the screen. Nevertheless, the computer still regards the command as one line.

After striking the RETURN key, the prompt in this example will look like:

```
This is IBM Personal Computer DOS Version  3.30.
The system time is 13:22:43.97.
The active subdirectory is C\.
At your service:_
```

If we have lost or forgotten the details of our latest invention, we can give the following command:

```
set <RET>
```

This results in a mysterious list like the following. It does, however, contain the formula:

```
PATH=
COMSPEC=C:\COMMAND.COM
PROMPT=This is $v.$_The system time now is $t.$_
The active subdirectory is $p.$_At your service:
```

We may create all kinds of artistic prompts. For instance:

```
prompt $n$g$g$g$g$g$g$g$g$g$g$g$g$g$g$g$g$g$g$g
$g$g$g$g$g$g $h$h$h$h$h$h$h$h$h$h$h$h$h$h$h$h$h$h
$h$h$h$h$h$h$h$h
```

One '$n', followed by twenty five times '$g', followed by a space and then by twenty five times '$h'. The dealer will be quite surprised!

By means of the list on the preceding page and by using our imagination, we can create all kinds of informative or curious prompts.

A RESET procedure or turning off the computer puts an end to the verbose prompt. In other words, a prompt we create ourselves lasts only as long as we are working with the machine. Obviously, we can save the formula in a batch program (see the previous pages).

However, if we get tired of a prompt we created ourselves, we do not have to do a RESET to get rid of it. We just type the bare form of the command:

```
At your service: prompt <RET>

C>_
```

9: THE BASIC FACTOR

Among the files on our system disk we usually find a file named BASIC.COM, BASICA.COM or GWBASIC.EXE. Strictly speaking, BASIC is not a system utility, but it is such a useful, and for some operations indispensable, instrument that we cannot avoid discussing it.

Whatever the name of the file in question, it contains the code of BASIC, which is a so-called programming language. This indicates a collection of commands that we can pass on to the computer in various ways and sequences. There are quite a few other programming languages, such as C, COBOL, and Pascal. However much they differ in amount, name and capacity of the commands, they have the same function: to provide building blocks for programs.

BASIC means Beginner's All-purpose Symbolic Instruction Code. It was developed between 1964 and 1970 under the supervision of the professors John G. Kemeny and Thomas E. Kurtz on the big 'mainframe' computers of Dartmouth College, the academic centre of the American state of New Hampshire.

Originally, BASIC was meant as a teaching aid for students who were someday going to work with real computer languages. But it was not long before paper tapes containing BASIC code (at the time there were hardly any disks or none at all) were accepted by other computers.

In 1975 the students Bill Gates and Paul Allen created a concise version of BASIC for what probably was the very first microcomputer - the MITS Altair. Both the computer and the MITS version of BASIC became big successes. Later the two teenagers founded their own company, which they called Microsoft. The company concentrated on the development of BASIC and other programming languages for various types of microcomputers, and it succeeded.

So, when IBM developed its Personal Computer and wanted to provide it, like the successful Apple II, with a BASIC version in ROM (see page 25), the firm approached Microsoft. The rest of the story can be found on page 10.

BASIC enables us to carry out procedures with the computer and the devices that the operating system is not capable of doing. There are commands, for instance, to perform calculations, and we can use it to control the printer, and lots more.

It is a problem, however, that there are dozens of versions of BASIC that differ from one another in details, and sometimes in essential points. This results from the fact that the original Dartmouth BASIC was developed in a government institution with public funds and therefore was 'public domain'. In other words, it belonged to no one and anyone could make use of it.

After Microsoft's success with a microcomputer-BASIC, numerous software companies tried to develop similar programming languages, resulting in a chaos of BASIC dialects. Even Kemeny and Kurtz have entered into business with their new version named True BASIC. But since most microcomputers contain a Microsoft version, it has in fact become the standard.

Our system disk also contains a Microsoft version of BASIC. In this chapter we will only discuss the most important built-in possibilities of BASIC, so that we in any case know how to formulate a command, how to load the system and how we can write a program ourselves.

Whoever wants to know the ins and outs of programming should buy a introductory book on BASIC and just get started.

9.1: Opening

We may be confronted with various versions of BASIC, depending on the make, the type and the version of our operating system.

First, there is the make. If we work with an IBM computer, we are confronted with three different types. The first is a version that is permanently stored in a part of the computer's memory called ROM (see page 25). This version is known as Cassette BASIC. To use it, we only have to turn on the computer, without a disk in the drive(s) (see page 15).

A message like the following may then appear on the screen:

```
The IBM Personal Computer BASIC
Version C3.20 Copyright IBM Corp. 1981, 1986

65536 Bytes free
Ok
```

The second version is known as Disk BASIC. It is included on the PC-DOS system disk in the form of a file named BASIC.COM. It contains a number of extra commands and facilities not included in Cassette BASIC, but for the rest the two versions are identical. In order to load Disk BASIC, we insert the system disk in drive A and type the following command:

```
A>basic <RET>
```

If all is well, we see a message like the following:

```
The IBM Personal Computer BASIC
Version D3.20 Copyright IBM Corp. 1981, 1986

62248 Bytes free
Ok
```

The third version offered by IBM is Advanced Disk BASIC, abbreviated to BASICA. Compared with plain BASIC, a number of facilities have been added such as the capability to process graphic images, but for the rest the two disk versions of BASIC are identical. To load BASICA, we type:

```
A>basica <RET>
```

This results in a message like:

```
The IBM Personal Computer BASIC
Version A3.20 Copyright IBM Corp. 1981, 1986

62248 Bytes free
Ok
```

Note that these BASIC versions only work on IBM computers, owing to the fact that they both need the 'Cassette BASIC' kernel in the IBM PC ROM.

Those who do not work with an IBM PC-DOS but with a version of MS-DOS, usually have a file named GWBASIC.EXE (reputedly from Gee Whiz BASIC) on the system disk at their disposal. Its contents are largely identical to IBM's BASICA. Assuming that the GWBASIC.EXE file is present on the disk in drive A, we load it as follows:

```
A>gwbasic <RET>
```

As a result, we see a message like the following:

```
GW-BASIC 3.20
(C) Copyright Microsoft 1983,1987

62195 Bytes free
Ok
```

In all versions of BASIC the term 'Ok' is the prompt, i.e. the invitation to pass on a BASIC command. Note that the cursor does not follow the prompt, but is on the next line.

A message like '60851 Bytes free' means that once the system and BASIC have been loaded, 60,851 bytes, that is just about 60 Kb, are available in the memory to load, write or process programs. This number is not related to the total capacity of the computer's memory. None of the BASIC versions is capable of processing more than 64 Kb of data at the same time.

9.2: Clearing the screen

It is sometimes useful to clear the screen and to place the cursor back in the top left-hand corner. To do so, we type the following command:

```
cls <RET>
```

With some versions of BASIC, we can achieve the same result by a ^<HOME> instruction. In other words, we keep the CRTL key down and then strike the HOME key.

9.3: Closing

If we have finished with BASIC and we want to return to the control of the operating system, we type:

```
system <RET>
```

As a result, the DOS prompt will appear on the screen again.

9.4: Contact with the screen

When using BASIC, much as when using the system (see page 16), we can put any character we like from the keyboard on the screen and pass it on to the computer by hitting the RETURN key. There is not much point in this, however, because only the reserved code words have any meaning. Anything else will be answered with the error message 'Syntax error'. For instance:

```
september <RET>
Syntax error
Ok
```

Assume we want the computer to print the word september on the screen. In that case we have to precede it by the command PRINT and type double quotation marks on either side of the word. Thus:

```
print "september" <RET>
september
Ok
```

Before we hit the RETURN key, we may correct any typing errors. Just as under system control, we can use the key on the right, showing an arrow pointing to the left, to remove the last character from that line.

To erase a whole line we strike the ESC key. And to cancel everything we have typed after the last 'Ok', we use the instruction ^C or ^<BREAK>.

Instead of the command PRINT, we may also type a question mark, as in:

```
? 'what day is it today?' <RET>
what day is it today?
Ok
```

Numerals do not need quotation marks.

We may also use the command to ask for the date or time:

```
? date$ <RET>
12-30-1988
? time$ <RET>
15:21:37
Ok
```

Note the dollar sign in the terms DATE$ and TIME$.

If we strike the RETURN key after the command PRINT or the question mark, a blank line will be the result:

```
? <RET>

Ok
```

9.5: The key bar

The bottom line of the screen usually shows the following:

```
1LIST   2RUN    3LOAD"  4SAVE"  5CONT   6,"LPT1 7TRON   8TROF
```

It indicates the functions assigned to the function keys under the control of BASIC. The significance they had under the control of the operating system (see page 22) no longer applies.

In cases where we do not want this bar displayed on the screen, we type the following command:

```
key off <RET>
Ok
```

To have the bar reappear on the screen, we type the reverse command:

```
key on <RET>
Ok
```

9.6: Contact with the printer

We send data to the printer in much the same way as we send data to the screen. But instead of PRINT we type LPRINT (for Line Printer). The question mark command is not permitted. An example:

```
lprint "28 december 1988" <RET>
Ok
```

After striking the RETURN key, only the prompt appears on the screen. The text we have typed between the quotation marks will be printed, assuming that the printer is properly connected to the computer and turned on.

Letters have to be enclosed between quotation marks. Numerals may be so enclosed but it is not necessary:

```
lprint 1988 <RET>
Ok
```

The maximum number of characters we can place on a single line with an LPRINT command is 132, but a normal-sized printer usually is not able to handle this number in its default mode of operation.

If we pass on an arithmetic calculation, the result will not appear on the screen, but on paper.

9.7: Calculating

Although BASIC is not designed for calculating, it is nevertheless capable of carrying out a number of arithmetic processes. The following symbols and their meanings are at our disposal:

^	- exponent sign
-	- negation sign
sqr	- square root sign
*	- multiplication sign
/	- division sign
+	- addition sign
-	- subtraction sign

In calculations the same order as in the above table is maintained. Thus, exponential functions take precedence over square root which takes precedence over multiplication, and so on. If another order of precedence is needed, we must use brackets to indicate which operation to carry out first.

We use the PRINT command we became acquainted with on page 158 to put the calculations on the screen. For instance:

```
print 123+456 <RET>
```

As soon as we release the RETURN key, BASIC produces the answer:

```
 579
Ok
```

We may also use the question mark instead of the PRINT command. It saves us some key strokes.

We shall provide a few examples without much comment. Addition:

```
? 123+456+789 <RET>
 1368
Ok
```

We may also use negative numerals. For instance:

```
? -123+(-456) <RET>
-579
```

Subtraction:

```
? 987-654 <RET>
 333
Ok
? 987-654-321 <RET>
 12
Ok
```

Multiplication:

```
? 123*456 <RET>
 56088
Ok
? 12*34*56 <RET>
 22848
```

Division:

```
? 987/654 <RET>
 1.509174
Ok
```

We must realise, however, that BASIC rounds off figures after the decimal point, so the answer could be inaccurate. As a test, we multiply the answer by the divisor in the above sum. Then we see:

```
? 1.509174*654 <RET>
 986.9998
Ok
```

Exponential:

```
? 12^2 <RET>
 144
Ok
? 12^3 <RET>
 1728
Ok
? 12^4 <RET>
 20736
Ok
? 12^5 <RET>
 248832
Ok
```

Square root:

```
? sqr(144) <RET>
 12
Ok
? sqr(12) <RET>
 3.464102
Ok
```

The need for brackets is illustrated by the following examples in which the same operations will be carried out with the same numbers:

```
? 987-654-321*5 <RET>
-1272
Ok
? (987-654-321)*5 <RET>
 60
```

9.8: Creating variables

We may be faced with numbers we do not need immediately, or which we want to put aside for a while. We can temporarily store these values - for as long as the computer is under control of BASIC or until we cancel the instruction - in a sort of parking place called a variable.

Variables are called into existence as soon as we enter their names or notations into the computer. We have to use letters for the name. Capitals are permitted but lower-case letters are easier to type. BASIC always converts them into capitals. No other symbols, no numerals and especially no quotation marks are allowed.

As soon as a variable is stored, we can call it up at any time for use in a calculation or in any other operation.

Suppose we want to store the numeral 28 in a variable we name x. In that case we type the following formula:

```
let x=28 <RET>
Ok
```

We may omit the term 'LET'. It saves typing three letters and a space. Thus:

```
y=15 <RET>
Ok
z=7 <RET>
Ok
```

We now have entered three variables, x, y and z, and stored the values 28, 15 and 7 in them. We can check that they have indeed been parked somewhere in the computer's memory just by asking:

```
? x <RET>
 28
Ok
? x+y+z <RET>
 50
Ok
? z-5 <RET>
 2
Ok
```

We are free to change the contents of a variable at will, either by replacing them by a completely different value, or by an arithmetic calculation on the original. For instance:

```
x=x+1 <RET>
Ok
? x <RET>
 29
Ok
```

9.9: The loop

Apart from just storing a value, there are several other possible uses for variables. We may, for instance, create a loop. A loop is a situation in which a variable gets a succession of different values until some prescribed condition is met.

Supppose we want to have all the numbers from 1 to 20 displayed on the screen. In that case we type a formula like the following:

```
for a=1 to 20 : print a : next a <RET>
```

In BASIC, a colon means a division between commands on the same line.

The first command creates the situation. We create a variable with the name 'A', store the value 1 in it, set up a counter that can give 'A' a succession of higher (or lower) values; and we determine that the final situation is reached as soon as 'A' has the value 20.

With the second command we tell the computer what to do every time 'A' gets a new value. In this case, print this value on the screen.

The command NEXT is inseparable from the FOR command. We use it to instruct the counter to give 'A' the next value.

If we want to increase the counter we created by means of the FOR command with another amount, for instance +2 instead of +1, we have to add a STEP instruction to the formula:

```
for a=1 to 20 step 2 : print a : next a <RET>
```

As a result, we see odd numbers only.

To reverse the order of the values of 'A', we make the first and last values change places. In addition, we give the STEP instruction a negative value, for instance -2:

```
for a=20 to 1 step -2 : print a : next a <RET>
```

Next, the numbers 20, 18, 16 etc. will appear on the screen.

To avoid the values of 'A' being printed on a new line, we may supplement the PRINT command with an extra symbol, like a comma:

```
for a=1 to 20 : print a, : next <RET>
```

As a result of the comma, BASIC displays the various numbers one after another, with an interspace of twelve positions.

We may also add a semi-colon to the PRINT command:

```
for a=1 to 20 : print a; : next <RET>
```

which results in an interspace of two positions.

9.10: Driving the printer

For normal printing work there is no need to make any adjustments to the printer, apart from some MODE commands (see page 122). We just have to turn it on and make sure that it has an adequate supply of paper. There is always a default setting.

However, printers are capable of much more. We have to give the proper instructions, as if it were a computer. In fact, it is a computer. A printer contains a chip that functions as a central processing unit.

Since the printer is provided with a large number of facilities, but not with its own operating system besides a few switches, we have to control the printer from the computer. In other words, we have to convert the text to be processed and printed into instructions for the printer. And immediately we run into difficulties. For a start, how do we pass on the instructions to the printer? And then there is the problem that a certain instruction is of significance to the one printer and not to the other.

Apart from some functions that are listed in the ASCII table (see page 68), there is no standardisation whatsoever. Nevertheless, let's try to find some consistency. First, we take another close look at the ASCII table and in particular the control instructions (see page 75).

BASIC enables us to pass on a control instruction, or any other symbol, directly to the screen or the printer. To do so, we have to use a specific function, in this case CHR$. This is the character function.

If we want to have the character corresponding with the decimal value 129 displayed on the screen, we type:

print chr$(129) <RET>

This turns out to be a ü. In order to check whether we can print this character on paper as well, we replace the term PRINT by LPRINT. Thus:

lprint chr$(129) <RET>

The one make or type of printer will produce the same character on paper, the other will not.

Let's return to the actual printer instructions. In principle, they can be divided into two types. The first one comprises a single symbol. This kind of instruction applies to all types of printer because it is included in the ASCII table, the only standard that has been recognised.

An example is the line feed instruction. It is included in the ASCII table under the decimal value 10. Therefore, we may also pass on this instruction as follows:

print chr$(10) <RET>

The second group of instructions consists of a combination of two or more values. This is where the standardisation lets us down. However, these combinations do have one thing in common. They all start with the ESC symbol, that is, the decimal ASCII value 27.

Take for instance the instruction code to print in italics. For IBM and Epson dot matrix printers it is 'ESC 4'. If we want to pass on this code, we have to formulate the command as follows:

lprint chr$(27)+"4" <RET>

The codes which are effective for a particular printer are usually listed somewhere on the back pages of the printer manual under 'Control codes' or 'ASCII codes'.

In any case, it would be wise to look them up and compare them with the ones below, which apply to the dot matrix printers of IBM and the Epson MX and FX types. If there are any differences, we simply replace them.

printer function:	ASCII code:	BASIC formula:
paper length of 11 inches (66 lines)	ESC C	CHR$(27)+"C"+"66"
12 inches (72)	ESC C	CHR$(27)+"C"+"72"
14 inches (84)	ESC C	CHR$(27)+"C"+"84"
prints 3 lines per inch	ESC A	CHR$(27)+"A"+"24"
6 lines per inch (default)	ESC 2	CHR$(27)+"2"
8 lines per inch	ESC 0	CHR$(27)+"0"
10 lines per inch	ESC 1	CHR$(27)+"1"
new line (line feed)	LF	CHR$(10)
vertical tab	VT	CHR$(11)
sets vertical tab at 16 lines (max.)	ESC B	CHR$(27)+"B"+"a"
horizontal tab	HT	CHR$(9)
set at 32 positions (max.)	ESC D	CHR$(27)+"D"+"a"
out-of-paper detector off	ESC 8	CHR$(27)+"8"
out-of-paper detector on (default)	ESC 9	CHR$(27)+"9"
fresh sheet of paper (form feed)	FF	CHR$(12)
returns print head to start of line	CR	CHR$(13)
condensed mode (132 char. per line)	SI	CHR$(15)
cancels condensed mode	DC2	CHR$(18)
enlarged mode (40 char. per line)	SO	CHR$(14)
cancels enlarged mode	DC4	CHR$(20)
double-strike mode (bold char.)	ESC G	CHR$(27)+"G"
cancels double-strike mode	ESC H	CHR$(27)+"H"
alternate mode (italics)	ESC 4	CHR$(27)+"4"
cancels alternate mode	ESC 5	CHR$(27)+"5"
emptying of the buffer	CAN	CHR$(24)
bell or buzzer	BEL	CHR$(7)
cancellation of all instructions	ESC @	CHR$(27)+"@"

9.11: Creating a program

So far we have only given the computer BASIC instructions, in other words, commands in the so-called direct mode. For instance:

```
print "it is now exactly "+time$ <RET>
```

A direct command is executed as soon as we hit the RETURN key, just as much as under the control of the operating system.

If we want to write a program file in BASIC, we have to number each line. In that way we tell the computer that the command should not be executed right away. Thus:

```
10 print "it is now exactly "+time$+" hours"
<RET>
```

To see the result of this program, we type the following:

```
run <RET>
```

and immediately we see:

```
"it is now exactly 5:29:00 hours"
Ok
```

We may also store this program as a file on a disk. Then we have to think of a name, for instance TIME. BASIC automatically adds .BAS as the extension. Assuming that the file has to be stored on the hard disk, the command is:

```
save "c:time",a <RET>
```

Note that we ended the formula with a comma and an 'a' (for ASCII). It ensures that the contents of the program file are stored in normal ASCII signs (see page 67) and not in the binary code.

In order to prevent BASIC from getting confused about lines or programs that are already in the computer's memory before we start writing, we should type the command NEW. Thus:

```
new <RET>
```

Let's write a program in which we instruct the printer to accept 12-inch paper (that is, 72 lines):

```
10 lprint chr$(27)+"c"+"72" <RET>
20 system <RET>
```

We call this program 12INCH and store it on the hard disk:

```
save "c:12inch",a <RET>
```

9.12: Loading and running a program

The way we load a BASIC program depends on the situation.
If we have addressed the program before and if the contents are still present in the computer's memory, the command RUN will be enough. Thus:

```
run <RET>
```

If a program is on a disk, like TIME (see the previous page), we can address it as follows:

```
Ok
load "time" <RET>
```

As a result, the program is stored in the computer's memory. Next, we give the command to run it:

```
Ok
run <RET>
```

We may combine these two commands as follows:

```
Ok
load "time",r <RET>
```

Note that we added a comma and a 'r' (for run) to the instruction.

The following command has the same result:

```
run "time" <RET>
```

What are we to do if we are under the system's control?
Assume that both BASIC and our 12INCH program are in the root directory of the hard disk, and that a printer is properly connected to the computer and turned on. The fastest way to load the 12INCH program is :

```
C>basic 12inch <RET>
```

Note that we omitted the quotation marks in the formula. They are not necessary if we address a BASIC program from the system.

In PC-DOS, we have to load BASICA instead of BASIC to be able to use graphic BASIC commands in our program. Thus:

```
C>basica 12inch <RET>
```

On the other hand, if we work with a version of MS-DOS instead of PC-DOS, so that we are dealing with GWBASIC, we have to type:

```
C>gwbasic 12inch <RET>
```

10: NEW OR VARIANT VERSIONS

As we have already mentioned on page 10, MS-DOS is in fact a collective name for a fairly large number of different versions of the operating system, which usually differ from each other with regard to certain facilities. Originally, MS-DOS was developed to the wishes of IBM, and that corporation added its own utilities as well, such as BACKUP, COMP, MODE, RESTORE and TREE; but IBM did not acquire the rights of ownership of the actual system. On the contrary, Microsoft reserved the right to place it at the disposal of other computer makers, and has indeed made use of this on a large scale.

Some manufacturers simply bought the standard package (in the jargon, 'generic MS-DOS'), others gave Microsoft instructions to include certain additions or changes, usually to receive copies of the IBM utilities mentioned above, and other manufacturers added utilities at their own initiative. The well-known manufacturer Wang, for instance, adds a version of MS-DOS containing a utility named SYSMODE to its computers, enabling the computer to switch from 'Wang mode' to 'industry standard mode' (which obviously refers to the way the IBM PC family operates). Otherwise, a software package like Lotus 1-2-3 will not function.

Now, let's take a look at the growth of PC-DOS, the IBM version of the operating system. The table on page 171 shows the increase in number and size of the various utilities. We can see that whenever a new version is put on the market, new utilities are introduced and the size of already existing files is significantly increased. Such an increase nearly always involves an extension of the programming code, as a result of which the utility in question offers us more facilities, or maybe an error in the design has been eliminated.

An example is the FORMAT utility. In its original form, the program was only capable of formatting single-sided disks to a storage capacity of 160 Kb, with the option of storing a copy of the system files on the disk in question.

The first revised version, which would eventually lead to PC-DOS version 1.10, also included the option to format double-sided disks, to a maximum capacity of 320 Kb. However, we still had the possibility of working with single-sided disks. For this purpose, the parameter /1 was added to the available options.

The second and third revised versions, resulting in the versions 2.00 and 2.10, added the possibility of providing disks with nine instead of eight sectors per track, so that the capacity of double-sided disks amounted to 360 Kb and that of single-sided disks to 180 Kb.

Moreover, the utility was now capable of formatting a hard disk. We only needed to specify the drive letter, usually a C, and after striking any key ('Strike any key when ready'), FORMAT would do the rest.

This specific use of the utility sometimes had (and still has) unforeseen consequences. Those who work with a hard disk usually include the system utilities on it. It may happen that a disk in drive A needs to be formatted and that, by accident, we have failed to state the drive letter (or the colon). Thus:

```
C>format <RET>
```

The result is:

```
Press any key to begin formatting drive C:
```

In this case, after striking any key, the hard disk, instead of the desired disk, will be completely formatted, and at the same time everything stored on it will be erased irrevocably.

This clear error in the design resulted in two reactions. The first came from the software designers and publishers Paul Mace and Peter Norton. They added an UNFORMAT command to their own range of utilities in which, in nearly all cases, an unintentional format procedure of the hard disk can be cancelled. Only the contents of the root directory will at all times be lost. It is therefore advisable to put no files in the root directory other than COMMAND.COM, CONFIG.SYS and AUTOEXEC.BAT.

The second reaction came from IBM and Microsoft themselves. On the introduction of version 3.00, the following warning will appear on the screen as soon as the hard disk is liable to be formatted:

```
WARNING, ALL DATA ON NON-REMOVABLE DISK
DRIVE C: WILL BE LOST!
Proceed with Format (Y/N)?
```

Moreover, striking any key is not enough to start the procedure, it has to be the RETURN key ('Strike ENTER when ready').

On the introduction of version 3.20, measures to protect a disk have become even more strict. First of all, the bare form of a FORMAT command is no longer accepted. If we fail to specify the letter of the drive in question, the procedure will be interrupted, showing the following message:

```
Drive letter must be specified
```

But even if we specify the drive in question properly, in this case drive C, this is not enough. We also have to state the label of the drive, if any:

```
Enter current Volume Label for drive C:_
```

If we do not enter the label or if we state the wrong label, the procedure will then be interrupted as well, for we see:

```
Invalid Volume ID
Format Failure

C>_
```

The introduction of the PC-DOS version 3.00 coincided with the introduction of the PC AT which is equipped with a disk drive capable of handling floppy disks with a storage capacity of 1.2 Mb.

In order to format a floppy disk in such a drive, we need not add a specific parameter to the FORMAT command. If we want to provide a disk in this drive with a different storage capacity, for instance the standard 360 Kb, we have to use the parameter /4 as we have already seen on page 40. Moreover, all the other parameters included in the system in the previous versions (/s, /v, /1 and /8) are still available.

10.1: Eight steps to PC-DOS 3.30

	1.00:	1.10:	2.00:	2.10:	3.00:	3.10:	3.20:	3.30:
IBMBIO.COM	1920	1920	4608	4736	8964	9564	16369	22100
IBMDOS.COM	6400	6400	17152	17024	27920	27760	28477	30159
COMMAND.COM	3231	4959	17664	17792	22042	23210	23791	25307
ANSI.SYS	-	-	1664	1664	1641	1651	1651	1678
APPEND.COM	-	-	-	-	-	-	-	5825
ASSIGN.COM	-	-	896	896	988	1509	1536	1561
ATTRIB.EXE	-	-	-	-	15123	15091	8247	9529
BACKUP.COM	-	-	3687	3687	5440	5577	6234	31913
CHKDSK.COM	1395	1720	6400	6400	9275	9435	9832	9850
COMP.COM	1620	1649	2523	2534	3471	3664	4184	4214
COUNTRY.SYS	-	-	-	-	-	-	-	11285
DEBUG.COM	6049	5999	11904	11904	15237	15552	15799	15897
DISKCOMP.COM	1124	1640	2074	2188	3752	4073	5792	5879
DISKCOPY.COM	1216	2008	2444	2576	4165	4329	6224	6295
DISPLAY.SYS	-	-	-	-	-	-	-	11290
DRIVER.SYS	-	-	-	-	-	-	1115	1196
EDLIN.COM	2392	2392	4608	4608	7183	7263	7508	7526
EXE2BIN.EXE	-	1280	1664	1664	2752	2816	3063	-
FASTOPEN.EXE	-	-	-	-	-	-	-	3919
FDISK.COM	-	-	6177	6369	8076	8173	8173	48216
FIND.EXE	-	-	5888	5888	6363	6403	6416	6434
FORMAT.COM	2560	3816	6016	6912	9015	9398	11135	11616
GRAFTABL.COM	-	-	-	-	1169	1169	1169	6128
GRAPHICS.COM	-	-	789	789	3111	3111	3220	3300
JOIN.EXE	-	-	-	-	-	15971	8955	8969
KEYB*.COM	-	-	-	-	10033	12051	15848	9056
KEYBOARD.SYS	-	-	-	-	-	-	-	19766
LABEL.COM	-	-	-	-	1260	1826	2346	2377
LINK.EXE	43264	41856	39936	39936	45970	38144	39076	-
MODE.COM	860	2509	3139	3139	5194	5295	6864	15487
MORE.COM	-	-	384	384	320	282	295	313
NLSFUNC.EXE	-	-	-	-	-	-	-	3060
PRINT.COM	-	-	4608	4608	7811	8291	8976	9026
PRINTER.SYS	-	-	-	-	-	-	-	13590
RECOVER.COM	-	-	2304	2304	4066	4050	4297	4299
REPLACE.EXE	-	-	-	-	-	-	11650	11775
RESTORE.COM	-	-	4003	4003	5413	5410	6012	34543
SELECT.COM	-	-	-	-	2079	2084	3826	4163
SHARE.EXE	-	-	-	-	8544	8304	8580	8608
SORT.EXE	-	-	1280	1408	1632	1664	1911	1977
SUBST.EXE	-	-	-	-	-	16611	9911	9909
SYS.COM	896	605	1408	1680	3629	3727	4620	4766
TREE.COM	-	-	1513	1513	2473	2831	3357	3571
VDISK.SYS	-	-	-	-	3080	3307	3307	3455
XCOPY.EXE	-	-	-	-	-	-	11200	11247

10.2: Sound and speed

As we explained on page 169, MS-DOS is in fact a collective name for numerous versions of the operating system, and these versions often differ from each other with regard to various facilities.

An example of an element that differs from manufacturer to manufacturer is the MODE utility. This utility in fact serves as a storage of all kinds of device drivers - in other words, programs to control certain parts of the computer. On pages 120 and 122 we have learned how to install several input and output channels, as well as how to use and change some of the functions of the screen.

In some versions of MS-DOS, including Compaq and Tandy, MODE also has various additional functions. We can use it to control the clicking sound we hear when tapping the keys. If we want to remove this clicking sound, for instance, we have to type the following command:

C>mode click=0 <RET>

In this formula, following the =, we may use other values varying from 0 (no sound at all) to 127 (maximum volume).

Nowadays, more and more computers are provided with a microprocessor which has a variable clock speed. This means that the speed at which data is being processed can be changed. How to give an instruction to change the speed usually depends on the arrangements that the manufacturer has made.

With Compaq and certain other brands, we can use the MODE command. For instance, the lowest speed at which a Compaq computer can function is 4.77 MHz, i.e. the pace of the original IBM PC. If we want the Compaq to slow down to this speed, we have to type the following formula:

C>mode speed=common <RET>

Other possibilities are 'fast' (7.14 or 8 MHz), 'high' (12 of 16 MHz), and 'auto' (automatically geared to the maximum speed possible under all circumstances).

To find out at which speed the computer is set at a given moment, we type the bare form of the mode command. Thus:

C>mode speed <RET>

Then we see, for instance:

SPEED=COMMON

By pressing the CTRL key, the ALT key and the \ key at the same time (or ^<ALT>\ for short), we can switch between 'common' and 'fast'.

We have to bear in mind, though, that the commands to control the sound and speed depend on the make and type of computer, and therefore function only on computers for which the version of the MODE utility is intended.

10.3: The keyboard (4)

With the introduction of version 3.30 of MS-DOS and PC-DOS we have to use a different method installing a non-American keyboard from the one that was described on page 74. The various .COM files beginning with the letters KEYB (KEYBUK.COM, KEYSP.COM, etc.) have been replaced by one file, named KEYB. COM. In addition to this file, we have to deal with a file named KEYB.SYS. These two files determine the effects of every keystroke and, as a result, replace the instructions in the ROM BIOS in this respect (see page 25).

The command to change the keyboard has to consist of the term KEYB, followed by the appropriate code number. This code number must be chosen from the following list:

country:	code:	country:	code:
United States	001	Great Britain	044
Canada (French) ...	002	Denmark	045
Latin America	003	Sweden	046
The Netherlands ...	031	Norway	047
Belgium	032	Germany	049
France	033	Australia	061
Spain	034	Finland	358
Italy	039	Arabic countries ..	785
Switzerland	041	Israel	972

Suppose we work with a British keyboard, i.e. a keyboard where the # sign on the 3-key has to make way for the pound sterling sign. Provided that both KEYB.COM and KEYB.SYS can be reached from the root directory, the command should be as follows:

```
C>keyb 044 <RET>
```

We may also add a second parameter to the KEYB command, namely a codenumber from the following list:

character set:	code:	character set:	code:
United States	001	Portugal	860
		Canada (French) ...	863
multilingual	850	Norway	865

For instance:

```
C>keyb 044,850 <RET>
```

As a result of this parameter, numerous characters with a decimal value of 155 and higher turn out to have been replaced in the extended character set (see page 75) by other characters.

Making some experiments with the ALT key and the numeric pad (see page 74) will make clear which characters it concerns.

10.4: Comparing (3) - FC

A very useful tool is the utility FC (for File Compare). It was developed by Microsoft and added to 'generic MS-DOS', but not adopted by IBM. Instead IBM developed the COMP program (see page 60). Anxious not to lose 'IBM compatibility', many computer manufacturers preferred COMP to FC when developing their own DOS version. Nevertheless, FC has a wider range of features, and it is more precise with regard to its explanations, as we will see below.

We put FC into operation by typing FC, whether or not followed by one or more parameters, plus the names of the two files to be compared. By way of experiment, let's return to the file we created in EDLIN (see page 102 and subsequently). Assume that we have two versions of this file by the names of MEMO.BAK and MEMO, both on a diskette in drive A, and that we want to check whether there are any differences between the two versions, and if so, what they are. To do so we type the following formula:

```
C>fc a:memo.bak a:memo <RET>
```

The result could be the following:

```
***** a:memo.bak
   A computer not supplied with a built-
in clock can have one fitted later. We
then have to buy an expansion card. This
card will usually also provide us with
***** a:memo
   A computer not supplied with a built-
in clock can have one fitted later. We
then have to buy an accessory known
as a clock card. This
card will usually also provide us with
*****

***** a:memo.bak
dealer may be able to give more details.
***** a:memo
dealer may be able to give more details.
*****

C>_
```

Note that FC places the pieces of texts in which the differences occur, under one another. The list ends by showing the last line of the two compared files.

It is possible to have FC number the lines, so that we know exactly where the differences in the files can be found. To do so, we have to add the parameter /n (for numbering) to the command. This parameter should precede the names of the files to be compared. So, in this example the formula should look like this:

```
C>fc /n a:memo.bak a:memo <RET>
```

Consequently, the result looks like:

```
***** a:memo.bak
   16:     A computer not supplied with a built-
   17:  in clock can have one fitted later. We
   18:  then have to buy an expansion card. This
   19:  card will usually also provide us with
***** a:memo
   16:     A computer not supplied with a built-
   17:  in clock can have one fitted later. We
   18:  then have to buy an accessory known
   19:  as a clock card. This
   20:  card will usually also provide us with
*****

***** a:memo.bak
   22:  dealer may be able to give more details.
   23:  ***** a:memo
   23:  dealer may be able to give more details.
   24:  *****
```

Note that, with this method, we can ascertain that the MEMO file consists of one more line than MEMO.BAK.

If we only want to see the first and the last line of the blocks of text containing the difference, we have to add the parameter /a (for abbreviate) to the formula. Thus:

```
C>fc /a /n a:memo.bak a:memo <RET>
```

Now the result is:

```
***** a:memo.bak
   16:     A computer not supplied with a built-
...
   19:  card will usually also provide us with
***** a:memo
   16:     A computer not supplied with a built-
...
   20:  card will usually also provide us with
*****

***** a:memo.bak
   22:  dealer may be able to give more details.
   23:  ***** a:memo
   23:  dealer may be able to give more details.
   24:  *****
```

Provided that the printer is connected properly to the computer, we can get the list of differences on paper by expanding the command as follows:

```
C>fc /a /n a:memo.bak a:memo >prn: <RET>
```

10.5: The interrupt function - BREAK

In certain circumstances, we may want the computer to react immediately to a ^C or ^<BREAK> instruction. In that case, we have to order the computer to check the keyboard more often to see whether we have typed such an instruction. As we have seen on page 127, this order consists of a BREAK=ON command in the CONFIG.SYS file. If we have not included such a command, we can still achieve the same effect from the system prompt by typing:

```
C>break on <RET>
```

To cancel this command, we just type:

```
C>break off <RET>
```

Should we have forgotten what the setting was, we can ask for it by typing:

```
C>break <RET>
```

The answer could be:

```
BREAK is off

C>_
```

10.6: Copying (2) - XCOPY

A very welcome addition to versions 3.20 and higher of MS-DOS and PC-DOS is the XCOPY utility. It can replace most of the COPY and BACKUP/RESTORE procedures, and it works considerably faster. Moreover, we may supplement an XCOPY command (like COPY, see page 55), with the parameter /v, to have the copied data verified.

Suppose we want to transfer a copy of all files in the active sub-directory to the disk in drive A, we type, assuming that the XCOPY.EXE file is in the root directory of the hard disk:

```
C>xcopy *.* a:/v <RET>
```

One of the differences from COPY is that first the following message will appear on the screen in order to confirm the command:

```
Reading source file(s)...
```

Just as with COPY, the names of the files in question will then appear on the screen one by one, followed by a message like:

12 File(s) copied

Unlike COPY, but just like BACKUP (see page 92), XCOPY is capable of making a selection from certain dates. For this purpose, we add the parameter /d (for 'date') to the command, followed by a colon and the date in question.

Suppose we want to transfer all files dating from December 29, 1988 or later to the disk in drive A. In that case we type the following formula:

C>xcopy *.* a:/d:12-29-88 <RET>

Which form the date in this command should get depends on the COUNTRY installation (see page 130).

We may combine the /v and /d parameters as in the following example:

C>xcopy *.* a:/v/d:12-29-88 <RET>

We can also instruct the system to ask for our permission for each file to be copied. To do so we add the parameter /p (for prompt) to the command:

C>xcopy *.* a:/p <RET>

Should we want to combine it with the parameters we used in the above examples, we type the following:

C>xcopy *.* a:/v/p/d:12-29-88 <RET>

As a result, we will get the usual confirmation first:

Reading source file(s)...

Next, the name of the first appropriate file will appear:

NOTES (Y/N)?_

It enables us to type a 'y' to confirm the command, or an 'n' if we do not want the file to be copied. After we have given an answer, the next file name will appear on the screen:

NOTES.BAK (Y/N)?_

And so on.

Also like BACKUP, but unlike COPY, a directory plus all its underlying sub-directories can be copied by adding the parameter /s (for sub-directories). To copy all files with a .DBF extension from the DBASE sub-directory and all sub-sub-directories to drive A, for instance, we must type:

C>xcopy \dbase*.dbf a:/s/v <RET>

Finally, we may also add the parameter /m (for modified) to the XCOPY command. In that case, like with BACKUP, only those files will be copied that have been altered since the last BACKUP or XCOPY procedure. Thus:

C>xcopy *.* a:/v/m <RET>

10.7: Replacing - REPLACE

The REPLACE procedure, introduced with version 3.20 of PC-DOS and MS-DOS, is related to the XCOPY procedure we became acquainted with on the previous pages. By means of this utility, we can automatically have one version of a specific file (or a group of files) replaced by another (e.g. more recent) version of the file(s) in question.

The procedure is as follows. Suppose we want to replace all files on the disk in drive A beginning with the letters NOT by versions with the same name in the active sub-directory. In that case we type the following command, assuming that the REPLACE.EXE file can be reached on drive C:

```
C>replace not*.* a: <RET>
```

The result may be the following:

```
 Replacing A:\NOTES.1

 Replacing A:\NOTES.2

 Replacing A:\NOTES.3

3 File(s) replaced

C>_
```

A number of practical parameters are available as well. If we add the parameter /a (for add) to the command, REPLACE will not only replace existing files but will also add new files to the indicated disk or sub-directory.

We may also add the parameter /d to the REPLACE command. In that case a file will only be replaced if the corresponding file on the source disk has a more recent date.

Furthermore, the REPLACE command may be followed by the parameter /p. As a result, the system will ask for our permission as each file is being tackled, like with the XCOPY procedure with the parameter /p. For instance:

```
REPLACE A:\NOTES.1 (Y/N)?_
```

If we add the parameter /r (for read-only), we get a REPLACE of any protected files as well (see page 65). Without this parameter the REPLACE procedure will be interrupted as soon as the system finds a protected file.

The parameter /s (for sub-directories) results in REPLACE searching all subdirectories on the target disk for any files that may apply to the command, and then replacing those.

Finally, the parameter /w (for wait) results in a pause before a file is added or replaced. It enables us to change disks regularly.

10.8: Expanding the path - APPEND

On page 90 we saw how to place a marker in the plethora of sub-directories by means of the PATH command. One of its disadvantages is that it can only be used for program files. In other words, files with the extension .BAT, .COM, or .EXE. With the introduction of versions 3.21 and 3.30 of the system, however, we can also have the system look for other types of files, including text files, namely by means of the APPEND utility.

As an example, suppose we want to use EDLIN, in the sub-directory DOS, to edit the text file NOTES in the sub-directory TEXTS. Without a PATH and/or an APPEND command in force, we have to formulate the command as follows:

```
C>\dos\edlin \texts\notes <RET>
```

Now we put up a signpost by means of the APPEND utility. Assuming it can be reached from the root directory, we type:

```
C>append \dos;\texts <RET>
```

Note that we use a semicolon for a division sign, just as with PATH.

As a result, we can formulate our instruction to the system more concisely:

```
C>edlin notes <RET>
```

We may also indicate another drive, for instance A or B, as the place where the system might find a file. In that case we type:

```
C>append \dos;\texts\;a:\ <RET>
```

Should we want to cancel an APPEND instruction, we have to re-issue the command, but this time followed by a space and a single semicolon only. Thus:

```
C>append ; <RET>
```

To check whether or not an APPEND instruction is already in force, we can type the bare form of the instruction. Thus:

```
C>append <RET>
```

If an APPEND instruction has already been given, the prompt will appear on the screen without any further message. If not, the reaction is:

```
No Append
```

The SET command we became acquainted with on page 153 will provide us with an alternative method. The result may be:

```
COMSPEC=C:\COMMAND.COM
APPEND=\DOS;\TEXTS;\A:
```

10.9: Duplicating (3) - BACKUP, RESTORE

With the introduction of version 3.30, the use of BACKUP has improved. For a start, it is no longer necessary to format a large number of disks before carrying out a BACKUP procedure. Now we can instruct BACKUP to use new floppy disks and format them at the same time. To do so, we have to expand the command with the parameter /f (for format).

As an example, assume we want to make a copy of the entire hard disk (drive C), using unformatted floppy disks. The command should then be:

```
C>backup c:\*.* a:/f/s <RET>
```

If we only want to copy, as on page 93, the files dated 20 December 1988 or later, we type:

```
C>backup c:\*.* a:/f/s/d:12-20-88 <RET>
```

In any case, we first see the usual instruction to insert the first disk in drive A. Immediately after we have hit any key, the FORMAT procedure will be carried out, ending with:

```
Format another (Y/N)?_
```

If we type an n, implying that, as far as we are concerned, the BACKUP program may be loaded, the usual messages appear on the screen:

```
*** Backing up files to drive A: ***
Diskette Number: 01
```

And so on.

Another improvement is that we can have BACKUP make a report of the state of affairs. In that case we have to add the parameter /l (for log, i.e. report) to the command. For instance:

```
C>backup c:\*.* a:/s/l <RET>
```

As soon as the procedure is put into operation, we first see a series of messages like the following:

```
Logging to file C:\BACKUP.LOG

Insert backup diskette 01 in drive A:
```

After the instruction has been carried out, the root directory of the hard disk (drive C) contains a file named BACKUP.LOG, consisting of a survey of all copied files, arranged in accordance with the numbers of the backup disks. If we want this result printed on paper, we type:

```
C>copy backup.log prn: <RET>
```

10.10: The 3.5-inch disk drive

With the introduction of first the IBM PC Convertible, and then the PS/2 range, a new type of disk drive was put on the market, namely the drive meant for the use of 3.5-inch disks. We have already mentioned them on page 26. If the manufacturer has equipped the computer with one or more 3.5-inch drives, we need not do anything special. Apart from the aspects of appearance and storage capacity, the use and operation of 3.5-inch drives are much the same as the normal or high capacity 5.25-inch drives.

The situation is quite different if we decide to install a 3.5-inch drive instead of, or in addition to, one or more normal disk drives, which in fact results in two different systems. We have to instruct the operating system to make a clear distinction between each drive, for instance, when carrying out a FORMAT or COPY procedure. Moreover, in that case we have to work with version 3.00 or higher. Older versions are not compatible with this.

Let's assume that drive A in our computer is a normal DSDD or HC drive (see page 27) and that in drive B, a normal 5.25-inch drive has been replaced by a 3.5-inch drive. To do this, there are special built-in assembly kits. If we work with version 3.00, 3.10, 3.20 or 3.21 of the system, we have to include the following instruction in the CONFIG.SYS file (see page 126):

```
drivparm=/d:1/f:2
```

If we work with version 3.30, 3.31 or higher, we are not allowed to use a DRIVPARM instruction. Instead, we have to include another instruction in the CONFIG.SYS file, namely the following, assuming that the DRIVER.SYS file can be reached from the root directory:

```
device=driver.sys /d:1 /f:2 /c
```

Or, depending on the capacity of the drive:

```
device=driver.sys /d:1 /f:3 /c
```

The parameter /d:1 in both formulas indicates that the instruction concerns drive B. 0 indicates drive A, 2 drive C, 3 drive D, and so on.

The parameter /f:2 lets the system know that we are dealing with a normal 3.5-inch drive with a capacity of 720 Kb, and /f:7 that we are dealing with a 3.5-inch drive with a capacity of 1.44 Mb. 0 is a normal 360-Kb drive, and 1 is a HC drive, that is, a PC AT-type drive with a capacity of 1.2 Mb.

The parameter /c (for check) is not necessary but quite useful, since it instructs the system to check, every time the door of the disk drive in question has been used, whether a swapping of disks has taken place.

After adding the instruction to the CONFIG.SYS file, we have to carry out a reset procedure or reload the computer. As a result, the system recognises drive B in its new role, so that we can carry out all the usual transactions with a disk in this drive.

11: ERROR MESSAGES

An offence against the rules of the operating system usually results in an error message on the screen. In some cases, the message is just a warning, leaving the proceedings uninterrupted. In other cases, the message prompts us to do something - to hit a key, for instance.

Because of the differences between the various versions of the system, it is not practical to present a complete survey of the error messages. The messages most likely to be met are summarised below.

* * *

`Access denied`

We tried to open a subdirectory as a file, or the file indicated is write-protected. We must first unprotect it.

`Allocation error, size adjusted`

A CHKDSK message. The problem has automatically been taken care of.

`Backup file sequence error`

We tried to restore a file that was backed up on more than one disk. We must first use the disk with the preceding part of the file.

`Bad command or file name`

The system does not recognise the file name or the command. Perhaps we have made a typing error, or maybe the file is not on the disk indicated.

`Bad or missing Command Interpreter`

The disk from which the system has been loaded of is missing, or does not contain a copy of the COMMAND.COM file. We must reload the system.

`Bad Partition Table`

The partition table of the hard disk does not have a DOS partition, or it is invalid. We must run FDISK, create a new partition table, and re-run FORMAT.

`Batch file missing`

The batch file that was being processed is missing. Perhaps we took away the disk it was on. In any case, the processing has been aborted.

`Cannot CHDIR to root`

A CHKDSK message. The root directory cannot be accessed. The disk is probably damaged.

`Cannot CHKDSK a SUBSTed or ASSIGNed drive`

A CHKDSK message. CHKDSK cannot access a subdirectory under a SUBST regime. We should undo the SUBST command and re-run CHKDSK.

`Cannot compare file to itself`

The indicated file names refer to the same file on the same disk. Perhaps we have forgotten to specify one of the drive letters.

`Cannot edit .BAK file--rename file`

EDLIN does not allow a .BAK file to be edited. Should it be necessary, however, then we must rename the file, or at least change the extension.

`Cannot execute FORMAT`

It is necessary to format the backup disk. But the FORMAT.COM file cannot be found in either the current directory or via the path we have set up.

`Cannot find system files`

The hidden system files cannot be found. We have to use another source disk to re-start the procedure.

`Cannot FORMAT nonremovable drive x`

The target in this BACKUP command with the /f parameter is a hard disk. To be able to use the hard disk as a target, we must first prepare it.

`Cannot load COMMAND, system halted`

The system had to reload COMMAND.COM, but there is too little memory, or COMMAND.COM cannot be found. We have to reset the system.

`Cannot perform a cyclic copy`

There is something wrong with the XCOPY command formula. Perhaps the source and the target directory are related to each other, causing a loop.

`Cannot RECOVER Entry, Processing Continued`

A CHKDSK message. The problem has automatically been taken care of.

`Cannot start COMMAND, exiting`

An attempt to load COMMAND.COM failed. Perhaps there is too little memory, or else we should add or adjust a FILES= command in the CONFIG.SYS file. We must now reset the system.

`Cannot use FASTOPEN for drive x`

Probably we typed a wrong drive letter. The FASTOPEN command can only be used on a hard disk.

```
Cannot XCOPY from a reserved device

Cannot XCOPY to a reserved device
```

Unlike COPY, the XCOPY command can only be used for transactions between disk drives.

```
CHDIR.. Failed Trying Alternate Method
```

A CHKDSK message. A technical problem occurred. We must reset the system and reload CHKDSK.

```
Configuration too large for memory
```

There is too little memory to load COMMAND.COM. We must adjust the FILES= or BUFFERS= commands in the CONFIG.SYS file and see what happens when reloading the system.

```
Content of destination lost before copy
```

The contents of one of the files to be used in a COPY command have been overwritten before the procedure had been completed.

```
Data error reading drive x
Abort, Retry, Fail?

Data error writing drive x
Abort, Retry, Fail?
```

We are confronted with the problem that data cannot be found, deciphered or stored on the disk in the drive indicated. Perhaps we have forgotten to insert the disk in the drive, or maybe we have failed to close the latch of the drive properly. In that case, we must correct the omission first and then type an 'r'. In all other cases we must type an 'a', after which the prompt will reappear and the computer's memory will have been emptied.

```
Directory is totally empty, no . or ..,
tree past this point not processed
```

No proper . and .. entries were found in the subdirectory. Probably at one time the updating process of the disk directory was interrupted. We must use the RECOVER utility to try and rescue as many files from the subdirectory as possible.

```
Disk boot failure
```

The operating system failed to load. If it fails a second time, we should use a copy of the system disk.

```
Disk error reading drive x
Abort, Retry, Fail?

Disk error writing drive x
Abort, Retry, Fail?
```

We are confronted with the problem that data cannot be found, deciphered or stored on the disk in the drive indicated. Perhaps we have forgotten to insert the disk in the drive, or maybe we have failed to close the latch of the drive properly. In that case, we must correct the omission first and then type an 'r'. In all other cases we must type an 'a', after which the prompt will reappear and the computer's memory will have been emptied.

```
Disk error reading FAT x

Disk error writing FAT x
```

A CHKDSK message. The File Allocation Table indicated is invalid. If both FAT 1 and FAT 2 appear to be invalid, the disk must be reformatted to make it usable again.

```
Disk full. Edits lost
```

There was not enough room on the disk we indicated to store the file. As a result, the revised version of the text was lost.

```
Disk not compatible
```

The FORMAT command cannot be executed because of the drive type.

```
Disk not Ready
```

No disk was found in the drive indicated. Perhaps we did not insert it properly, or we forgot to close the latch.

```
Disk unsuitable for system disk
```

Bad sectors were found in the tracks that have been reserved for the system. The system files will therefore not be copied.

```
Diskette is not a backup diskette
```

The BACKUP or RESTORE procedure cannot be executed, since the contents of the disk were not created during a BACKUP procedure.

```
Diskette is not last backup diskette
```

The BACKUP or RESTORE procedure cannot be executed, since the contents of the disk are not the last of a BACKUP series.

```
Diskette not compatible
```

The target disk is not formatted in the same way as the source disk. The procedure cannot be executed.

```
Diskette not initialized
```

The CHKDSK program has not succeeded in finding the FAT or the directory. Presumably, there is something wrong with the disk. It is advisable to try and copy the files first.

```
Divide overflow
```

There must be an error in the command, in the system or in the computer causing the command in question to be aborted.

```
Drive not compatible
```

The target drive type differs from the source drive type. The procedure can not be executed.

```
Duplicate file name or file not found
```

The file has not been found on the disk indicated, or the name we wanted to give to the file already exists.

```
Entry error
```

We have made an error in typing an EDLIN command or entering text.

```
Entry has a bad allocation error, size adjusted
```

A CHKDSK message. The File Allocation Table contains an invalid sector number. The file in question has been truncated at the end of the last valid sector number.

```
Entry has a bad attribute
```

A CHKDSK message. If the command is given with the /f parameter, CHKDSK will try and correct the error.

```
Error loading operating system
```

The system failed to boot from the hard disk. We must put a system disk in drive A and have the system loaded from there.

```
Error opening log file
```

Perhaps we made an error in indicating the drive or path, or else there is no more room in the directory we indicated.

```
Error reading partition table
```

```
Error writing partition table
```

A technical error occurred on the hard disk. We must run FDISK again.

```
Errors found, F parameter not specified
Corrections will not be written to disk
```

A CHKDSK message. Some errors haven been found, but nothing will be done about it until we re-run CHKDSK with the /f parameter,

```
Errors on list device indicate that
it may be off-line. Please check
```

The printer is not (or not properly) connected or not switched on.

```
EXEC failure
```

An error was encountered. We must adjust the FILES= command in the CONFIG. SYS file.

```
FASTOPEN already installed
```

The FASTOPEN utility has already been loaded. No further action is required.

```
File allocation table bad, drive x
Abort, Retry, Ignore?
```

The File Allocation Table is defective. In order to control and possibly to repair it we must type an 'a' and issue a CHKDSK command with the /f parameter.

```
File cannot be copied onto itself
          0 File(s) copied
```

Apparently we have given the command to copy a file and to store the copy under the same name on the same disk. This is not possible. We must either give the copy another name or store it on another disk.

```
File creation error
```

We have included an invalid character in the file name, or there is no more room in the directory. We should run CHKDSK to find out what is wrong.

```
File not found
```

The file named in the command formula has not been found in the specified drive. Perhaps we have made a typing error, or maybe have failed to indicate the correct drive or (sub-)directory.

```
File is cross-linked on cluster xx
```

Two files seem to share a cluster of data. We must copy these files by means of a COPY or XCOPY command to another disk, and then erase the originals.

```
File is READ-ONLY
```

We cannot write to the file indicated. We have to undo its read-only status first.

```
File name must be specified
```

We failed to indicate the file that we want to edit.

```
File not found
```

The file indicated could not be found. Perhaps we made a typing error, or we put the wrong disk in the drive.

`File not in print queue`

There is no file with that name in the queue. Perhaps we made a typing error.

`FIND: Invalid number of parameters`

We have forgotten to include all the necessary information in the FIND command formula. Perhaps we failed to type the characters we wanted to be found.

`FIND: Syntax error`

Owing to an error in the FIND command formula, it cannot be executed. Perhaps we have forgotten to type one or more inverted commas.

`First cluster number is invalid, entry truncated`

A CHKDSK message. The problem has automatically been taken care of.

`Fixed backup device n: is full`

The target disk is full.

`Format failure`

An unrecoverable error has been found on the disk during the FORMAT procedure. We must reissue the command. If the message reappears, we may as well discard the disk, as it can no longer be used.

`General failure`

An error for which no other error message exists has been found. Perhaps the disk type and drive type do not match, or the disk is not properly formatted.

`Has invalid cluster, file truncated.`

A CHKDSK message. The problem has automatically been taken care of.

`Illegal device name`

The MODE command cannot be executed in this form. Perhaps we made a typing error.

`Illegal parameter`

The specified parameter of the command or the indication of the file is not correct.

`Incompatible drive types`

The DISKCOPY command cannot be executed. Perhaps we have a disk drive with an insufficient capacity.

```
Incompatible system size
```

The transfer of the system files cannot take place since there is not enough room on the target disk. We must format a blank disk instead and then copy the files from the original disk.

```
Incorrect DOS version
```

We cannot use a system utility from another system version than the one we loaded.

```
Incorrect number of parameters
```

We made an error in the number or the form of the parameters. We must retype the command.

```
Incorrect parameter
```

The parameter we added was not recognised. Perhaps we made a typing error.

```
Insert disk with \COMMAND.COM disk in drive x
and strike any key when ready
```

The COMMAND.COM file could not be found. We must insert a disk containing this file in the drive indicated.

```
Insert disk with batch file
and strike any key when ready
```

We have to put the disk containing the batch file back in the drive, before the rest of the commands can be carried out.

```
Insert DOS disk in drive x
and strike any key when ready

Insert system disk in drive x
and strike any key when ready
```

The hidden system files could not be found. We must insert a system disk in the drive indicated.

```
Insufficient disk space
```

The disk in question has not enough free storage space to contain the new file. We have to remove one or more existing files first.

```
Insufficient memory
```

There is not enough memory available to load the program file(s) indicated. Perhaps we can discard the RAM-disk, if one is installed.

```
Insufficient memory for system transfer
```

There is not enough storage room on the disk.

```
Insufficient room in root directory
Erase files from root and repeat CHKDSK
```

CHKDSK has found so many remainders of files that the root directory is full. We have to remove or erase some files first.

```
Intermediate file error during pipe
```

The procedure cannot be executed. Perhaps one of the files named in the formula is not present or there is too little room on the disk to contain the temporary files.

```
Invalid baud rate specified
```

The first number in the MODE command is not correct. Only 110, 150 300, 1200, 4800, 9600, and (in DOS 3.30 and up) 19200 are permitted, or at least the first two figures of those rates.

```
Invalid characters in volume label
```

We have broken the rules of typing the name of the label in the formula. We have probably used an invalid character.

```
Invalid COMMAND.COM in drive x
```

The COMMAND.COM file that the system tried to reload is from another DOS version.

```
Invalid current directory
```

A CHKDSK message. The problem has automatically been taken care of.

```
Invalid date
```

The form in which the date has been typed is not correct. We may have made an error in the order (month/day/year), or used the wrong division sign. Only '-' and '/' are allowed.

```
Invalid device
```

We have not specified the channel or destination properly. Maybe we have typed a colon or a numeral in the wrong place or included more than one space after the actual command.

```
Invalid directory
```

The directory indicated does not exist. Perhaps we made a typing error.

```
Invalid disk change
```

We changed a disk when one or more files were still in use. We must re-insert the original disk.

```
Invalid drive specification
```

The drive is not specified properly. Perhaps we forgot to type a drive letter, or a colon.

```
Invalid language specified
```

The language code in the KEYB command is not valid. Perhaps we made a typing error.

```
Invalid media or track 0 bad - disk unusable
```

FORMAT was unable to format track 0 of the specified disk. Perhaps we used the wrong parameter. If not, we must discard the disk and use another one.

```
Invalid number of parameters
```

We made an error in the number or the form of the parameters. We must retype the command.

```
Invalid parameter

Invalid parameters
```

The parameter we added was not recognised. Perhaps we made a typing error. If there were two or more, we may have typed them in the wrong order.

```
Invalid partition table
```

While trying to load DOS from the hard disk, the system found invalid information in the partition table. We must use FDISK to try and correct the problem.

```
Invalid path, not directory,
or directory not empty
```

The specified or active (sub-)directory cannot be removed because it is not empty, or because the path command contains an invalid name or character.

```
Invalid path or file name
```

A file name or (sub-)directory in the COPY command does not exist. Perhaps we have forgotten to indicate the relevant drive.

```
Invalid path or parameter
```

We made an error in the APPEND command. The /x and /e parameter can only be specified alone, and only the first time APPEND is loaded.

```
Invalid subdirectory
```

A CHKDSK message. With the /f parameter, CHKDSK will try and correct it.

```
Invalid subdirectory entry.
Tree past this point not processed.
Convert directory to file (Y/N)?
```

CHKDSK has found an entry error. We should return to the system level, to try and copy all the files of this subdirectory. Next, we must re-run CHKDSK with the /f parameter, and give 'y' for an answer.

```
Invalid switch
```

A '/' was found in the DEVICE=VDISK command, though not followed by an 'e'.

```
Invalid syntax
```

The formula is not correct. Perhaps we have made a typing error.

```
Invalid time
```

The form in which the time has been typed is not correct. We may have typed an invalid division sign instead of a colon.

```
Invalid Volume Label
```

The text we typed does not match the volume label of the disk.

```
KEYB has not been installed
```

The message says it all.

```
*** Last file not backed up ***
```

The target disk was filled to capacity while the file was being backed up.

```
List output is not assigned to a device
```

Perhaps the printer is not (or not properly) connected, or not switched on.

```
Memory allocation error
Cannot load COMMAND, system halted
```

A technical error has occurred. We have to reload the system.

```
Missing file name
```

We have failed to type the old or new name of the file.

```
Missing operating system
```

The hard disk does not contain the system files. We must load the system from a disk in drive A and FORMAT the hard disk with the /s parameter.

```
Must specify destination number
```

We probably forgot to specify a line number.

```
Must specify COM1, COM2, COM3 or COM4
```

The message is clear enough.

```
Must specify ON or OFF
```

We have probably made a typing error. We have to issue the command again, either in its bare form, or followed by ON or OFF.

```
No files found
```

On account of the criteria, no files were found to REPLACE.

```
No fixed disks present
```

The FDISK program did not detect a hard disk. Perhaps it was not properly installed.

```
No operating system on fixed disk
```

There are no system files present on the hard disk.

```
No paper
```

The printer is either not turned on, or out of paper. We must turn the printer on and press the On Line-switch, or add paper and retry.

```
No room for system on destination disk
```

No room has been reserved for the system files on the destination disk. The SYS procedure cannot be carried out.

```
No room in directory for file
```

The directory on the drive indicated is filled to capacity. The changes we made to the file are lost.

```
No room in root directory
```

The volume label could not be created, probably because there was not enough room left in the root directory. We must delete a file and try again.

```
Non-DOS disk
```

The File Allocation Table appears to contain invalid information. We must re-format the disk.

```
Non-System disk or disk error
Replace and strike any key when ready
```

We have failed to load the system. There may be something wrong with the hard disk. Or, if we work with a floppy disk, perhaps the disk does not contain any system files or we have not used the right system disk. Maybe we should try another system disk.

```
No source drive specified
```

We must type the command again, this time including source and target drive.

```
No system on default drive
```

No hidden files were found on the disk in the active drive.

```
No target drive specified
```

We must type the command again, this time including source and target drive.

```
Not enough room to merge the entire file
```

Because of insufficient memory, EDLIN's T command was not able to merge all of the specified file.

```
Not ready error reading drive x
Abort, Retry, Fail?

Not ready error writing drive x
Abort, Retry, Fail?
```

We are confronted with the problem that data cannot be found, deciphered or stored on the disk in the drive indicated. Perhaps we have forgotten to insert the disk in the drive, or maybe we have failed to close the latch of the drive properly. In that case, we must correct the omission first and then type an 'r'. In all other cases we must type an 'a', after which the prompt will reappear and the computer's memory will have been emptied.

```
Out of paper
```

Either the printer is not properly connected or it has run out of paper, or the output channel is not properly installed.

```
Parameter not compatible with fixed disk
```

A parameter has been added to the command which cannot possibly be meant for the hard disk, such as /1 or /8.

```
Path name too long
```

The length of a path may not exceed 63 characters.

```
Print queue is empty
```

There is nothing wrong, but there are currently no files being processed.

```
Print queue is full
```

The maximum number of files (10) has been reached.

```
Printer error
```

Either the printer is not properly connected, or it has run out of paper.

```
Probable non-DOS disk.
Continue (Y/N) ?_
```

A CHKDSK message. There are difficulties reading the floppy or hard disk. Perhaps it has not been formatted properly, if at all, or it is faulty.

```
Program too big to fit in memory
```

The file to be loaded is bigger than the amount of memory available. Perhaps we can discard the RAM-disk, if one is installed.

```
Requested drive is not available
```

The drive specified does not exist.

```
Restore file sequence error
```

We did not insert the backup disks in the correct order.

```
Sector not found reading drive x
Abort, Retry, Fail?
```

```
Sector not found writing drive x
Abort, Retry, Fail?
```

Some data cannot be found, deciphered or stored on the disk in the drive indicated. First we type an 'r', to repeat the command. If it fails, we may type an 'f', which causes the system to try and skip the problem area.

```
Source diskette bad or incompatible
```

The DISCOPY procedure was aborted. Either some bad sectors were found while reading the source disk, or the disk is not compatible with the drive type.

```
Source does not contain backup files
```

The files on the disk were not created by a BACKUP procedure.

```
Specified drive does not exist,
or is non-removable
```

No valid drive letter was indicated. Perhaps we made a typing error.

```
Target cannot be used for backup
```

No files could be created on the target disk. We must try another one.

```
Target diskette may be unusable
```

The target disk may be irreparably damaged. In order to control and possibly to repair it, we have to issue a CHKDSK command.

```
Target diskette write protected
Correct, then strike any key
```

The target floppy disk is write-protected. We must either remove the protecting tab or use another disk.

```
Target is full
```

There is no more room on athe target disk. We must either delete some files, or use another disk with more storage room.

```
Track 0 bad-disk unusable
```

During a format procedure, one or more bad sectors have been found on track 0 of the disk. This disk can no longer be used.

```
Tree past this point not processed
```

Some error has occurred. The procedure cannot be completed past this point.

```
Unable to create directory
```

The subdirectory specified cannot be created. There may already be a subdirectory with this name or maybe the root directory is full.

```
Unrecognized command in CONFIG.SYS
```

The CONFIG.SYS file contains an invalid command. Perhaps we have made a typing error. We have to recreate the file first and then reload the system.

```
Unrecoverable read error on drive x
Track a, side b

Unrecoverable read error on source
Track a, side b

Unrecoverable read error on target
Track a, side b

Unrecoverable verify error on target
Track a, side b

Unrecoverable write error on target
Track a, side b
```

An unrecoverable error has been found on the specified place of the source or the target disk. The procedure will be continued but data may be missing.

```
Warning-directory full
```

The root directory is filled to its maximum. The RECOVER procedure will be aborted.

```
Warning! Diskette is out of sequence
Replace the diskette or continue
Strike any key when ready
```

We have made an error in the order of the BACKUP disks. There is now an opportunity to insert the right disk in the drive. The RESTORE procedure will resume as soon as we press a key.

INDEX OF COMMANDS

In this index of commands, anything between square brackets indicates an alternative or a facultative addition to the command in question; "x:" and "y:" mean the designation of a drive; "{text}" stands for a string of characters or a certain formula; "{file}" for a file or a group of files; "{name}" for the name of a sub-directory; and "{channel}" for an input or output channel.

INDEX OF NAMES AND CONCEPTS